Computer simulation of an anti-air operation in the Combat Information Center.

Mamou, Jawad

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

COMPUTER SIMULATION OF AN ANTI-AIR OPERATION
IN THE COMBAT INFORMATION CENTER

by

Jawad Mamou

December 1986

Thesis Advisor: R. E. Ball

ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable)
7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Computer Simulation of an Anti-air Operation in the Combat Information Center

12 PERSONAL AUTHOR(S)
Mamou, Jawad

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 December	110

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A computer program simulating an anti-air operation conducted from the C.I.C. of a ship was written in the C language to run on an MSDOS personal computer. This program simulates the main functions of a C.I.C. and incprporates into the NPS interactive simulation of an engagement at sea (ISEAS) with a radar module and a weapons module developed by others. The contribution of this thesis to ISEAS is weapons direction, decision-making, and graphics display of the tactical information in a useful form.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ OTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Robert E. Ball	22b TELEPHONE (Include Area Code) 408-646-2885 / 22c OFFICE SYMBOL 67Bp

D FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Computer Simulation of an Anti-air Operation In The Combat Information Center

by

Jawad Mamou
Ensign, Royal Moroccan Navy
B.S., Ecole Royale Navale, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

ABSTRACT

A computer program simulating an anti-air operation conducted from the C.I.C. of a ship was written in the C language to run on an MSDOS personal computer. This program simulates the main functions of a C.I.C. and incorporates into the NPS interactive simulation of an engagement at sea (ISEAS) with a radar module and a weapons module developed by others. The contribution of this thesis to ISEAS is weapons direction, decision-making, and graphics display of the tactical information in a useful form.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I wish to thank my thesis advisor, Professor Robert E. Ball, for his guidance, Captain Nakagawa, Chairholder, Tactical Analysis Chair, for the long hours he spent sharing his knowledge of tactical analysis, my sponsors, Major and Mrs. John Sapienza, and Mr. and Mrs. Robert Tarozzi for their caring and their friendship. I would also like acknowledge the Royal Moroccan Navy for giving me the opportunity to attend the Naval Postgraduate School.

I. INTRODUCTION

A. BACKGROUND

The stream of data generated during an anti-air warfare (AAW) operation at sea can be overwhelming, and an efficient and complete analysis of the information provided by all the different shipboard sensors will take more time than available. One solution to the problem is the design of a computer based Decision Support System (DSS). This system will have the capabilities of today's computer technology, especially graphics, so that it will be able to utilize information gathered on own ship, on enemy forces, and also historical and intelligence data. The system will effectively predict enemy near term moves, will organize all of this collected information, and will present it in a form (pictures) that will be easy to use and understand. With a sophisticated algorithm, it can lead to appropriate decision making.

One objective of this thesis is to design a small interactive DSS and to show examples of a few modules written to deal with various features of the engagement of an incoming threat. After analyzing the available information about this target and about our weapons, an optimized use of our power against this target will be suggested to the user. An analysis of his action will be carried out and

the results displayed. Another objective is to show examples of how to convert some of the raw information provided by some of the ship borne sensors to a comprehensive picture on the screen of an IBM PC/AT computer. The scope of this thesis will be limited to an AAW operation conducted from the Combat Information Center (CIC). This work does not provide an operational capability because it is meant to be educational. Most of the elements used here are generic since the real ones are classified.

B. ISEAS

The Interactive Simulation of an Engagement At Sea (ISEAS) is a PC based computer program, under developement at the Naval Postgraduate School, where many of the features of a real engagement are simulated. It is written in the C language and reproduces the functions of a CIC, from the use of the radars to the direction of the weapons.

This paper has two main parts. The first part (Chapters II, III and IV) deals with the tactical situation and the decision-making. Thus, a CIC doctrine is simulated given the information provided by the RADAR module and the WEAPONS module which are written by other students. The second part (Chapters V and VI) will look at some of the ways to display information with the aid of graphics.

II. THE BIG PICTURE

A. DEFINITIONS

What happens when there is a target to engage? How is the engagement decided, and when? What happens during the engagement, and afterwards? What is the role of the Combat Information Center? How does it conduct this task? This chapter will introduce the user of ISEAS to the different events that occur during the engagement and to the different possible outcomes. Also, the user will be made familiar with the structure of ISEAS and be able to identify the different modules and understand their relationship to each other.

A big picture of the engagement will be drawn here giving a clear idea of how an outcome can be reached. To do so, an Essential Elements Analysis is conducted, and a relationship between a wanted outcome and an end-event is established.

First, the definitions of some keywords used throughout this thesis are needed. Usually, these keywords mean different things because of the variety of the contexts and also because the tactical doctrines keep changing with time. The main difference is their scope and extent of use. The author was unable to find any specific written definitions. But here they are defined and the mean the following.

Combat Information Center (CIC): it is the space within the ship in which the battle is fought. Tactical control of

all the ship's combat systems resides in the CIC. Inter-ship communications and other command and control functions are also performed in the CIC. Specifically, the following five functions are performed:

1. Information collection
2. Information processing
3. Information display
4. Information dissemination
5. Weapons direction (involving the fire control system)

Fire Control System (FCS): it is the set of computers, consoles, data buses interfacing with sensors, weapons launchers and weapons direction equipment. It takes inputs from the sensors, processes them in the computers and generates firing solution computations and weapons direction.

Command and Control unit: it is the processor of information that generates solutions other than the firing computations, such as information about a detected target, pulled out from a large database system; or a suggestion of an optimised use of the ship's weapons against a raid. Such suggestions are based on a set of rules of different tactical situations and doctrines, implemented in the algorithms used by the processor.

B. ESSENTIAL ELEMENTS ANALYSIS

In a one-on-one engagement between a ship and an aircraft or a missile there are four possible outcomes:

1. The target is killed, and the ship survives
2. The target survives, and the ship is killed
3. The target survives, and the ship also survives
4. Both the target and the ship are killed

An Essential Elements Analysis (EEA) can be conducted for each outcome to see what are the events that lead to it. However, since we are more interested in looking at the action of the ship against the attacker, and since we assume (at the present time) that our ship is invulnerable, an EEA for the target kill alone will be conducted.

There are two possible outcomes for the target:

1. Target killed
2. Target not killed

A target is killed if the ship succeeds in countering it and denying its penetration at one of the defense envelopes (long range, medium range or short range missile envelope); or if the target succeeds in entering all the envelopes but fails to function accurately and misses the ship. Again since our concern is to actually prevent a successful enemy attack, we will look at the first possibility; the ship effectively counters the target. For the ship to succeed in killing the target the following events should occur successfully:

- target detected in time

- target identified and designated correctly as enemy
- target assigned to the fire control system
- target acquired by the fire control system
- target tracked accurately
- a fire control solution generated correctly
- a proper weapon is available and ready for use
- missile is launched
- missile intercepts the target
- target is killed by missile warhead

The algorithm of the engagement of a target by a weapon will follow these straightforward steps, provided that each step is accomplished. An assessed probability of happening can be computed or determined from a table prior to the actual engagement. The ultimate result is a value of how successful the missile was in killing the target or the total probability of kill given a single shot (Pkss).

An analysis of the final outcome "target killed" is useful for determining the sequential steps for the algorithm of engagement.

The analysis of the final outcome "target not killed" is useful in pointing out the events that might cause a failure in killing the target. The elements and events that lead to these two final outcomes should be considered if we want our model to be more realistic. In the real world, these elements and events can be countless. In our model they are limited to a few obvious ones, easy to understand and varied enough to cover different areas.

A target not killed means that it was able to fly through all three envelopes successfully. An EEA should be conducted for each envelope, but since they differ only slightly from each other, only one EEA will be done for the long range envelope. The differences from the other two envelopes will be mentioned at the end of this chapter.

The target may not be killed for the two following reasons:

- the ship fires and misses
- the ship does not fire at all

If the ship fires and misses it will be for the following reasons:

- missile malfunction (versus a failure to function) given a good fire control solution
- fire control system malfunctions

One category of missile malfunction could be a hardware problem, leading to a technical failure of the missile's guidance unit or the propulsion unit. Another category could be the guidance algorithms or the software was not sophisticated enough to prevent the missile from being diverted away from the target or simply an unsuitable navigation law.

The fire control system malfunction can lead to a bad fire solution. For example, inaccurate parameters or wrong information about the target can be the cause. If there is a tracker problem or a successful use of the ECM by the enemy,

the target will not be tracked properly and a good solution will then be difficult to get. Another malfunction of the fire control system can be caused by a bad human-machine interface; the operator can be too quick or too slow to react, or he can push the wrong button, or becomes confused and misinterprets information. The consequence can be an inappropriate choice of the missile, such as a too slow missile for a fast or highly maneuverable target, or an inappropriate choice of the time of launch that may lead to a target out of range after firing the missile. Finally, a mechanical failure of a switch or a light to indicate a state (not ready or target not acquired, etc.) can be the source of a malfunction.

If the ship does not fire it could be for the following reasons. A fire control solution is generated, but the launcher fails to function or is not in a proper state or position. Also, a Command and Control problem may prevent the firing despite the availability of a solution. A safety factor or a management problem could be the source.

If a fire control solution is not generated, the ship cannot fire. This can be caused by several different events:

- target not detected or detected too late because of an ESM malfunction or sensors malfunction or inadequate performances
- target detected but not identified/designated properly because of the IFF malfunction or failure
- target designated correctly but not assigned/acquired by the fire control system, and that can be caused by a

communication problem between the sensors and the FCS. A defective manual assign/acquire phase can be caused by the delay of the operator

- target acquired but not tracked because of a tracker failure or an evasive action taken by the target (lost track) etc...

Figure II-1 shows the relationship of these events.

The main difference between the long range envelope and the medium range envelope is the detection-designation phase. Because that was accomplished in the previous envelope and needs to be done only once, there is no need for its repetition. The same applies for the short range envelope.

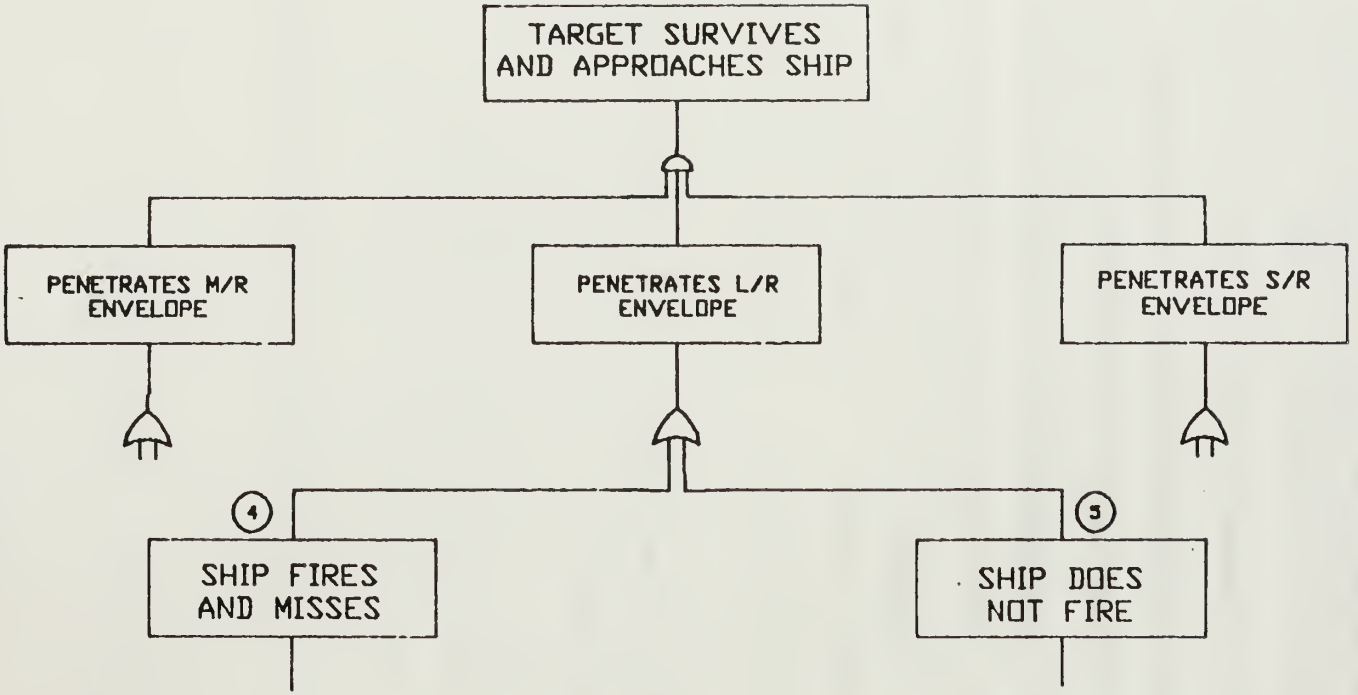

Figure II-1. Fault tree of the end-event "Target survives".

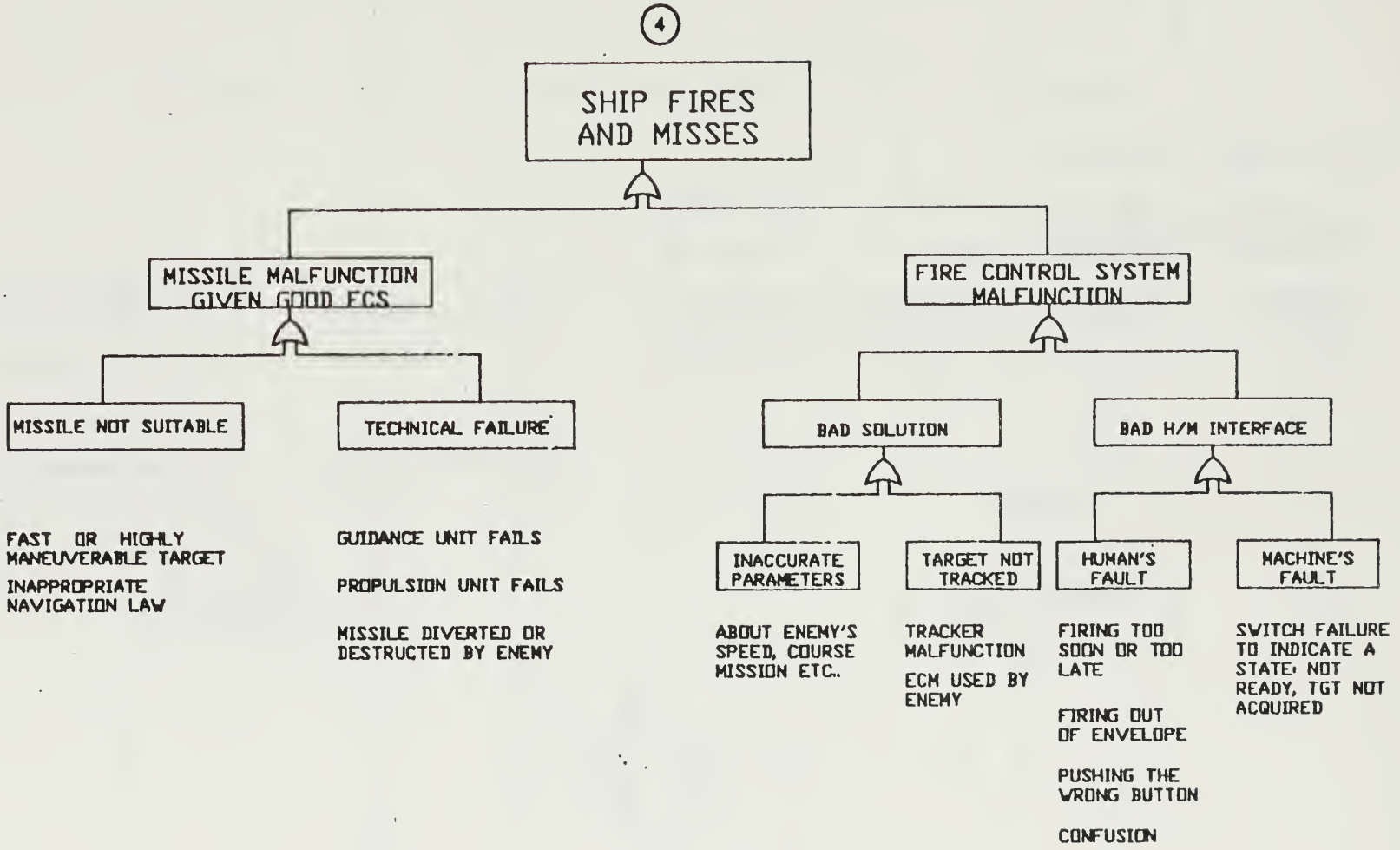

Figure II-1. Fault tree of the end-event "Target survives". (Continued)

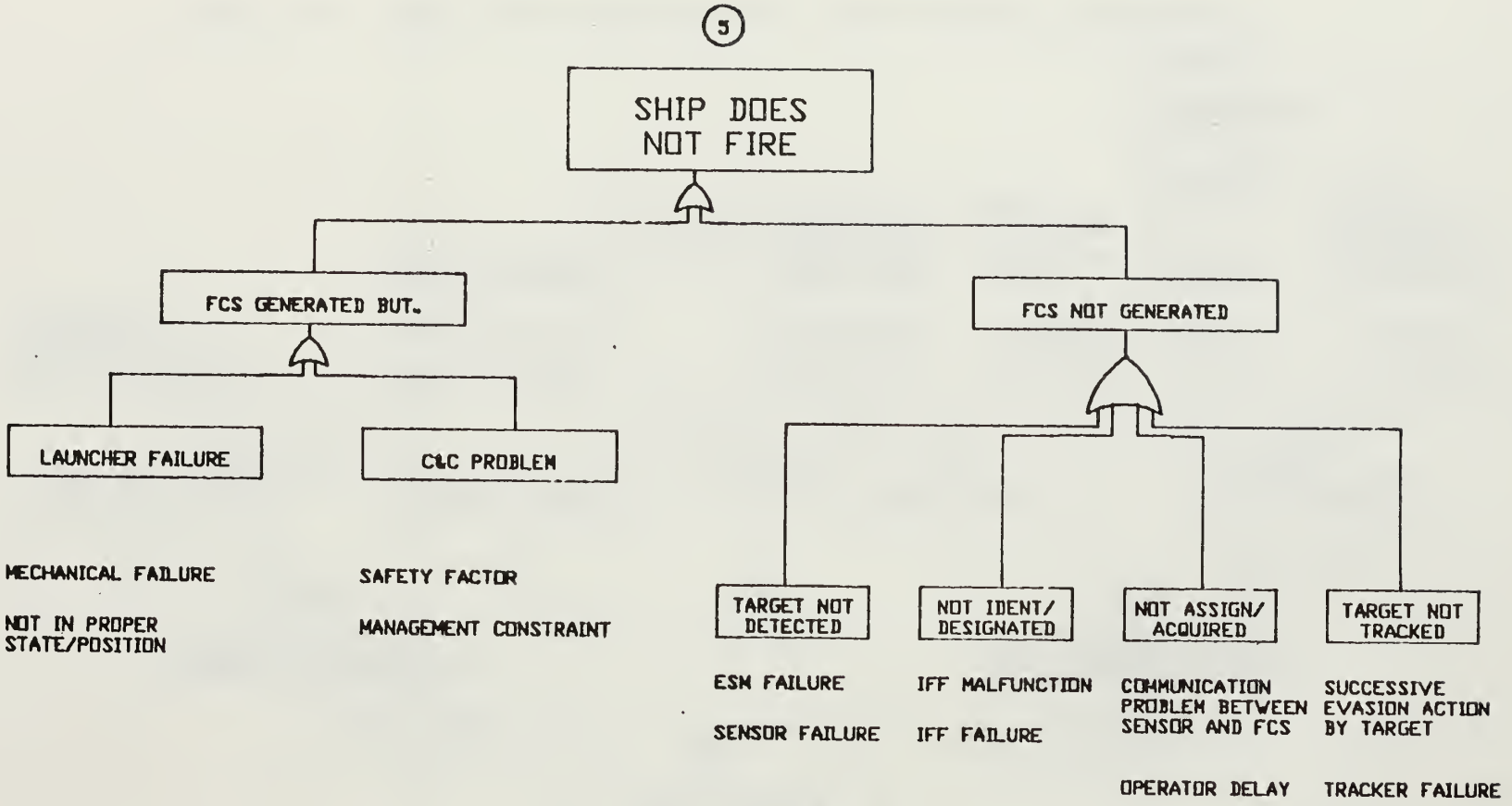

Figure II-1. Fault tree of the end-event "Target survives". (Continued)

III. DESCRIPTION OF ISEAS

A. BRIEF DESCRIPTION

ISEAS is an interactive simulation of an engagement between a ship and an attacking aircraft. It simulates the functions carried on in the Combat Information Center, from the search phase to the eventual kill of the target. Three modules have been written separately by NPS students at the time of this thesis. These three modules have been combined by the author into one large program. These modules are the radar module dealing with the detection and the tracking phases; the weapons module, simulating the flyout of a missile from the launch to the intercept point; and finally the Combat Information Center module which solves the firepower equation, computes the geometric parameters of the engagement, and processes the information about the target and displays it in text and graphics modes.

Each module is written in the C language and uses two types of variables. Local variables are used exclusively by their module where they are declared. Global variables are exchanged between modules so they need to be declared outside of the modules. In, this way, they are available for every module that needs them. Typical global variables are the target's position X, Y and Z.

These three modules are explained briefly below, pointing out their outputs and how they work with each other.

B. RADAR MODULE

The radar module [Ref. 1] is the first one called by the program when the simulation starts. Logically, it is the first phase of an engagement; the target has to be detected if any engagement is to occur. This module simulates the major functions of a radar, from the generation of an illuminating beam to the energy losses, to the detection of an object. The weather, any jamming by the enemy, and the characteristics and limitations of the radar are taken into account. This module simulates two different types of radars; the search radar and the tracking radar. The input for this module is the actual position of the target, which at this time is simply on a straight and level flight path. The output is the position of the target as tracked by a radar: contaminated by the uncertainities and approximations because of the present technology.

C. WEAPONS MODULE

This module [Ref. 2] becomes active when the firing of a missile is ordered from the CIC. After the order, it reads the present position of the target and its predicted intercept point solved for by the CIC module. Then it calls the launcher function where the initial azimuth and elevation of the launcher are computed. Following is the call for the flyout

function. In this function the target's position is read every time it iterates, then the atmospheric conditions (density and temperature) are computed. After that the lift slope curve coefficient is calculated in the preparation for the solving of the angle of attack. The missile's thrust and weight are computed next. Finally, the guidance function is called and the angle of attack is solved. The output of the flyout function is the updated missile's velocity and position. The updated position of the missile is sent to the CIC module where it is processed and displayed.

D. COMBAT INFORMATION CENTER MODULE

This module is the controller of the program. Basically, it performs three different tasks. The first one is to solve the firepower equation, producing the firing geometry and analyzing it. The second one is to "orchestre" the different modules and subroutines of ISEAS on a time step event basis. The third one is to gather the information available and display it in a useful way. For details of how these tasks are carried out refer to Chapter IV in this thesis.

E. FLOW OF THE PROGRAM

Figure III-1 shows the flow of the program. First, the input program is run and the variables are either changed or the default values are kept. These variables are saved in disk files. Then the simulation is run. It asks for which data files to be loaded. Then, the simulation starts by displaying

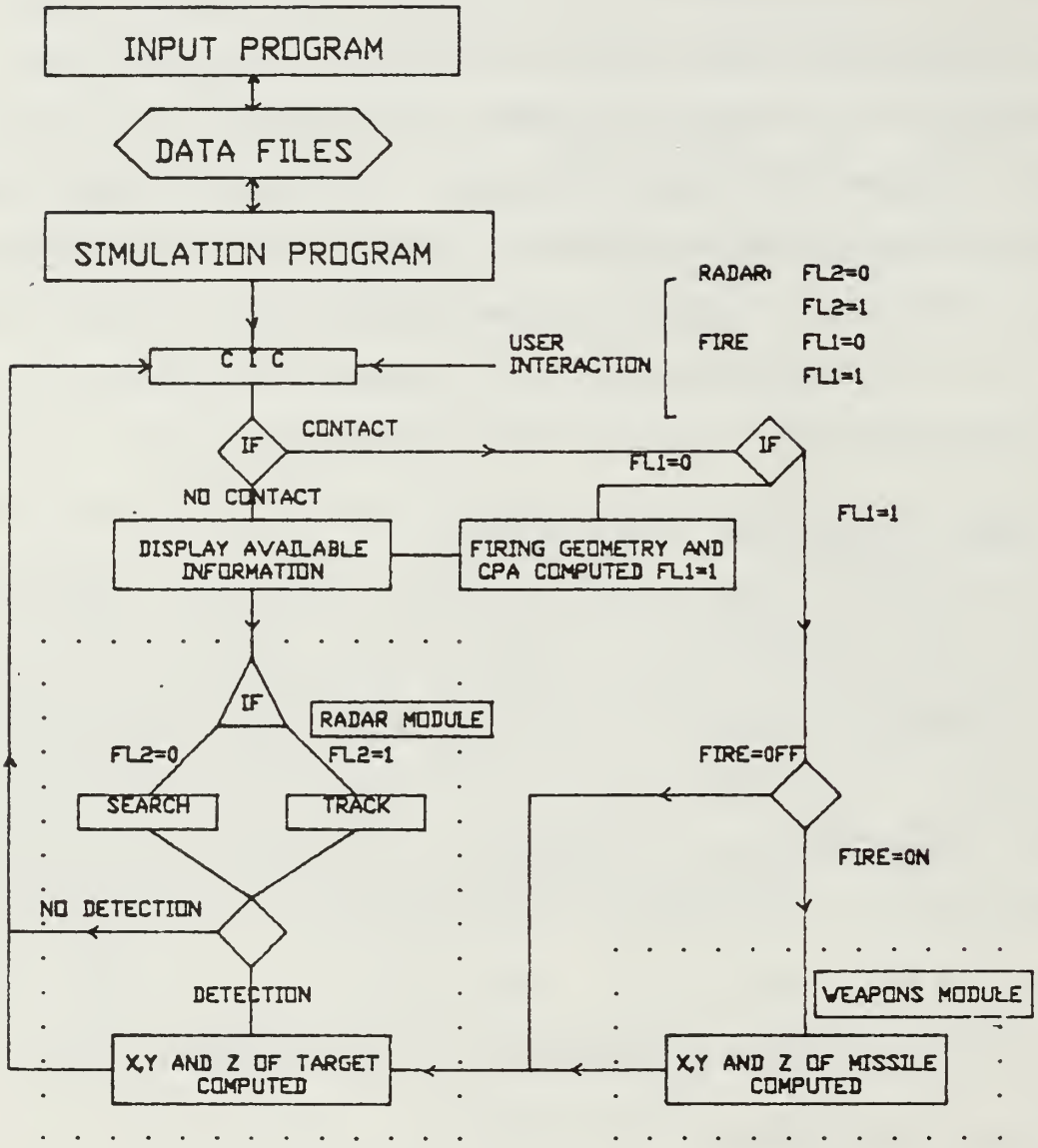

Figure III-1. Flow of the Program

the tactical screen with the information on own ship only. The radar module is called and the search radar is activated. The search for an object starts into a loop. The detection of the target turns on a flag whithin the radar module when it reaches the maximum detection range. The later is computed after the technical characteristics of the radar and the weather conditions are checked. Once the detection flag is turned on, the target is reported to the CIC module where it is identified and designated as hostile. The firing geometry solved by the firepower equation and the closest point of approach (CPA) module is drawn on the tactical screen. Thus, up to now, we have a tactical screen where the maximum detection range, and the long, medium and short ranges of the missiles are drawn. On the 'text' screen, information concerning the radar being used and the weapons available are displayed. Also, information about the newly detected target is available. The CPA and its location are computed and drawn on the tactical screen along with the predicted flight path of the target. The position of the target is updated by continuously calling the radar module. This position is displayed on the 'text' screen and plotted on the tactical screen as it becomes available. Now, the user is looking at a radar-type screen with the necessary information permitting him to make a decision as to when to fire and which weapon to use. The CIC module, by this time, has already computed the possible firing sequence of the missiles depending on the

range of the target from the ship. The individual probability of kill given a single shot (Pkss) for each firing, and the total probability of kill are also available. Thus, the user knows what are his chances to kill that target. The plotting of the target's position continues as long as it is within the detection range and the user doesn't take action. When he decides to fire at the target, the tracking radar is turned on by calling the tracking module. The later takes over the position updating with better accuracy. The user assigns the target to a weapon (long, medium and short range missiles provided they are available) and when he decides to fire, the weapons module is called, and the launcher position is computed and a ready signal is sent back to the CIC module. Now, the CIC module will be calling the tracking function, within the radar module, getting the instantaneous position of the target, displaying it and passing it to the weapons module. The latter goes to its flyout function and "flies" the missile accordingly and returns the missile'sposition to the CIC, which plots it on the same screen. This sequence is followed until the missile stops flying, then an analysis of the firing is done. The result is displayed in the form of a Pkss number and a total probability of kill if previous missiles were fired at that target. If the user decides to fire another missile, a delay time is observed, and the weapons module is called again in the same manner as before. The user keeps firing as he wishes provided some weapon is

available and ready to be fired, and as long as the target is within the missile's envelope. Once the target is killed (which is determined by a kill threshold value), or when it leaves all envelopes, a summary of the engagement is displayed as to how many missiles were fired and when, their intercept range and their Pkss. A total probability of kill is also displayed . The CIC module calls the radar module, setting the flags to their initial values, and activating the search radar for a new target.

This simulation goes on until the user decides to stop it or until he expends all his missiles.

IV. THE DECISION MAKING

A. DISCUSSION

This chapter deals with the operational side; the decision-making and the analysis of the user's action. To define the steps needed for this task, a tactical situation is developed pointing out the events happening during an anti-air operation.

TACTICAL SITUATION: The tactical situation may be best developed by listing the sequence of events which lead to the interception of the target by the surface to air missile. The following events would normally be expected to occur in sequence:

1. DETECTION: The attacking aircraft is detected in some manner. The detection will be conducted from the ship with the surveillance radar.

2. IDENTIFICATION: The target must be identified as friend or foe. If the target is identified as enemy, an alert must be given.

3. EVALUATION: Information as to position and altitude, direction of motion, and probable type of attack is needed in order to direct the specific means of interception that may be best employed at least risk to the ship.

4. ASSIGNMENT: After the target has been evaluated, the the user will assign a weapon to it with the assistance of the command and control unit and the fire control system.

After receiving the information needed from the search radar and the tracking radar, and also from the weapons unit, the Command and Control unit will have to determine an optimized weapon assignment. The Fire Control system will generate a solution suitable to the target, and the Command and Control unit, given the tactical and management requirements, will decide on the engagement parameters, so the fire order will be issued at the right moment with the appropriate conditions.

As is seen in the previous paragraph, two different units are involved so far; the Command and Control unit, and the Fire Control system.

B. COMMAND AND CONTROL UNIT

Every element in the Combat Information Center, from the sensors to the weapons, performs differently in the various situations presented. For example a missile may have a better probability of hit when the target is closer. A sensor performs better when it is not functioning in a hostile environment. A target is easier to counter when the appropriate weapon is readily available and so on. The level of performance can be measured as a percentage of a maximum capability. Another way to think of it is how much can you expect that element to perform correctly. Usually, this performance is related to one or more parameters, such as a range, velocity, availability etc.. The value, which is a probability number, is either fixed or variable, found in a

table, guessed, or computed from a function. In this situation, a decision is more difficult to make because the decision maker is not judging if that element will function or not. Rather he will have to evaluate its performance with a value ranging from zero to one. Furthermore, a complete action involves many different elements. Consequently, the overall evaluation of the mission's result is more difficult to determine. The Command and Control unit plays a vital role in this process. With the appropriate set of rules for computing probabilities and finding expectations and sophisticated tactics and algorithms, the Command and Control unit can provide the decision maker with an optimized reaction in any tactical situation. The advantage of the Command and Control unit, which is merely a processor loaded with a sophisticated program, is speed and memory. It is built to deal with various tactical situations, to interact with the different sensors and equipment on board, to process information according to a set of rules at very high speed, and to recommend an optimal action in a easy way to use. Usually, the output is expressed by means of graphics on tactical CRT screens.

The Command and Control unit implemented in ISEAS gives a simple example of how a number of missiles will be suggested to fire at given times so they will achieve a wanted total kill probability. Also it gives the individual and the accumulated kill probabilities for each missile fired or to be fired.

Another use of the Command and Control unit is to help manage the missiles regarding the availability and the kind of mission and the type of target to deal with. There will be missiles on board for different ranges (long, medium and short) with different capabilities; mainly speed and navigation law. Once the information about the target is available, the Command and Control unit will help in choosing the right weapon for the target. The problem of managing the number of missiles could be solved by the same unit. The operator can either fire at the target every time it is possible regardless of how many missiles will be expanded, or wait for the target to get closer so the expected kill probability reaches a preestablished value and then fire. In the later case less missiles would be fired and the Command and Control unit is appropriate for such computations.

The Command and Control unit module in ISEAS can be modified and improved to deal with more complex problems and take into account more parameters.

The first problem that this unit solves in this program is the performance of the missile measured against the intercept range. A missile is supposed to have a maximum range and a minimum range. In this model, the missile will have a high probability of kill at the minimum range (RN) and a low probability of kill at the maximum range (RX) and a zero probability outside its operating range (either beyond the maximum range or within the minimum range). The probability

of kill given a single shot (Pkss) will be a function of the intercept range, varying between two nominal values within the operating range. The simplest type of function that will do this task is a linear function (Figure IV-1). The function will have the following form:

$$Pkss(range) = A * range + B \quad ; RN < range < RX$$

$$Pkss(range) = 0 \quad \text{if} \quad range > RX \quad \text{or} \quad range < RN$$

This function can be used in two different ways; to compute the kill probability given an intercept range, or find the range at which a specific probability of kill can be achieved.

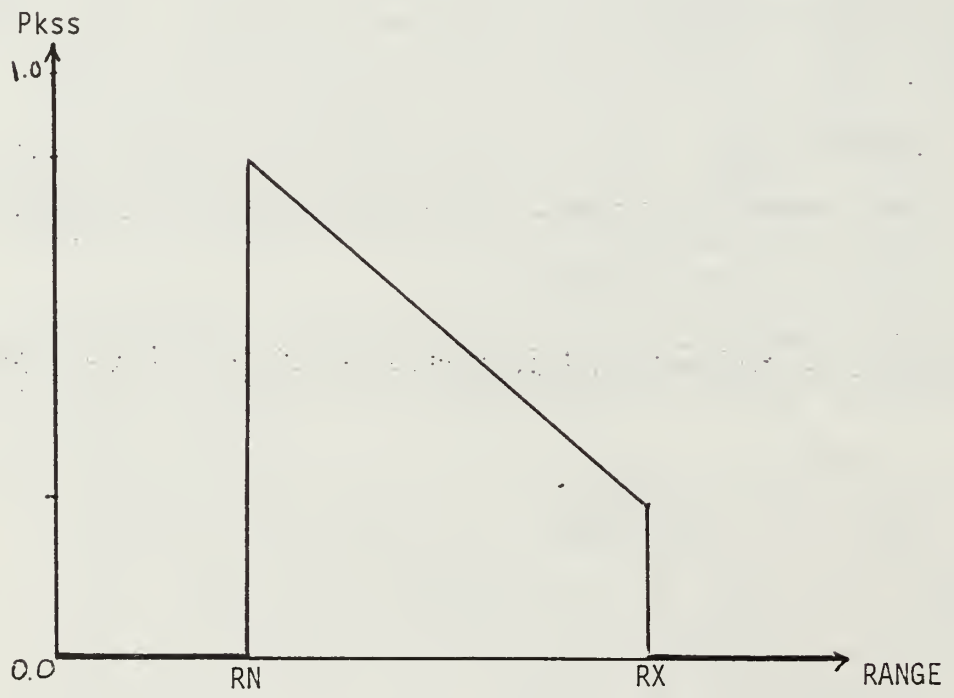

Figure IV-1. Linear function of Pkss vs. Range

This function gives the probability of kill given a single shot, which is considered a success p versus a failure q. The accumulated probability of kill for multiple shots is given in the following form:

Pk(total) = 1 - q ** n

with q = 1 - Pkss for each missile fired.

But since the value of Pkss varies for each missile (because each missile is fired at a different distance) the formula for the total probability of kill will be

Pk(total) = 1 - (1 - Pkss(i)) where Pkss(i) is the probability of target kill for the ith missile fired.

Another use of the function of the probability of kill is to determine the intercept range at which a given value of the kill probability is sought.

This is done by inverting the initial function to the following

range = (Pk(range) - B) / A .

This range is the intercept range. For the intercept to occur at this range the fire control system has to solve the geometry and find the firing range which is prior to the intercept range.

C. FIRE CONTROL SYSTEM

One of the main functions of the fire control system is to determine how many missiles can be fired at an incoming threat. The solution is given by the FIREPOWER equation

derived here. The raid could be either an aircraft or a missile, with the following conditions:

- the raid is moving at a constant velocity, constant direction and constant altitude (non maneuvering-target)
- the heading is not necessarily toward the ship
- the geometry and the equations are treated in three dimensions
- the missile has a maximum intercept range and a minimum firing range

To fire the maximum number of missiles, the first missile must be fired so that it will intercept exactly at its maximum range. Subsequently missiles must be fired as fast as possible, until the raid has closed within the minimum firing range or left the envelope. In developing this equation, a shoot-look-shoot policy has been used with a variable look delay time.

The shoot-look-shoot model is especially valid when only one fire control radar or guidance system is available for that particular target. To solve the firepower equation, the closest point of approach (CPA) is needed. A module called CPA solves this problem. Once the CPA is solved for, the geometry for the firepower equation can be drawn and solved.

1. The Closest Point of Approach Module

The variables used in this module are:

CPA : the closest point of approach in the X-Y plan.

TCPA : the closest point of approach in the X-Y-Z space.

XT,YT,ZT : the initial position of the target.

VT,CT : the target's velocity and course.

$\dot{X}T,\dot{Y}T$: the target's velocity components.

XS,YS : the initial position of the ship.

VS,CS : the ship's velocity and course.

$\dot{X}S,\dot{Y}S$: the ship's velocity components.

$\dot{X},\dot{Y}$: the components of the target's relative velocity.

The output will be the true TCPA (three dimensions). The geometry is shown in Figure IV-2. Zero degree heading is considered along the X-axis and positive to the left (Cartesian coordinates).

The ship to target horizontal distance is computed knowing the initial positions.

$$R = SQR\ [(XS-XT)**2 + (YS-YT)**2]$$

The angle A from the target to the ship in the horizontal plan is given by

$$A = ARCTAN\ [(YT-YS) / (XT-XS)]$$

The components of the target's relative velocity are computed next. The ship is taken as the reference so that

$$\dot{X} = \dot{X}T - \dot{X}S$$

$$\dot{Y} = \dot{Y}T - \dot{Y}S$$

where

$$\dot{X}S = VS * \cos(CS)$$

$$\dot{Y}S = VS * \sin(CS)$$

$$\dot{X}T = VT * \cos(CT)$$

$$\dot{Y}T = VT * \sin(CT)$$

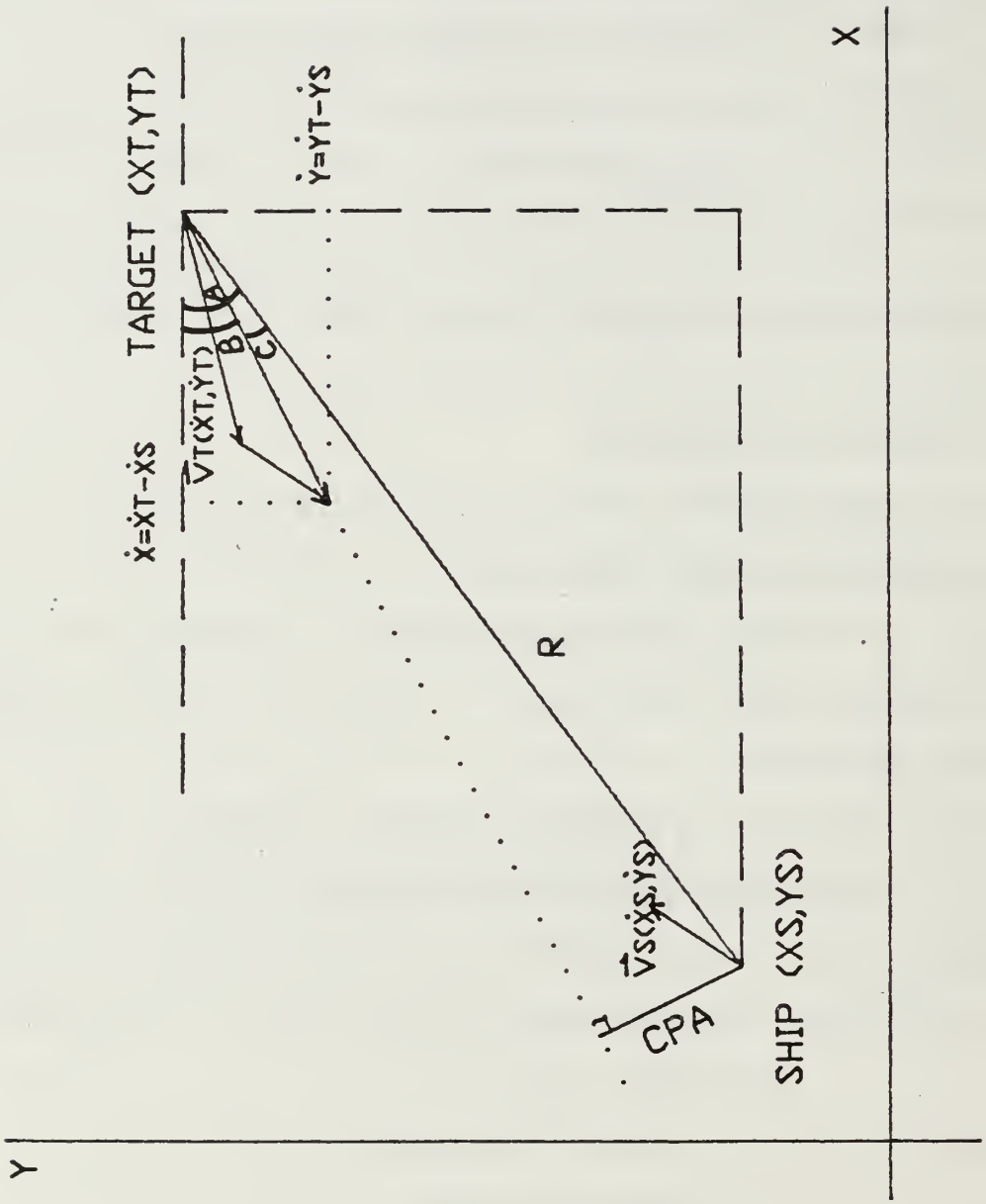

Figure IV-2. Closest Point of Approach

The angle of the relative velocity is found from

$$B = \text{ARCTAN}\ (Y/X)$$

A new angle C is computed from the geometric difference of the two previous angles. This angle gives the geometry of the final triangle that will lead to the CPA in the X-Y plan.

$$C = A - B$$

The CPA in the X-Y plan is the opposite side of the angle C. Thus,

$$CPA = R * \sin(C)$$

The true CPA takes the altitude of the target into account.

$$TCPA = SQR\ [(CPA)**2 + (ZT)**2]$$

TCPA will be referred to as the CPA meaning the true CPA.

A test is made on the angle C. If it is larger than 90 degrees (absolute value) no CPA case exists. Geometrically speaking, the target will not get any closer than where it is at the time of the computation.

2. The Firepower Equation Module

Now that we have the CPA we can construct the geometry for the engagement. For the purpose of clear demonstration two different geometries will be considered:

CASE I: the target is on any heading (variable CPA), but with no delay look time (T = 0). [Ref. 3]

CASE II: this is the general case, the target is on any heading, with a variable delay look time T.

The following terms, in addition to the ones already defined in the CPA module will be used in the derivation:

SAM : surface to air missile.

RX : maximum intercept range of SAM.

RN : minimum intercept range of SAM.

VM : speed of SAM.

t : time.

n : number of missiles after initial intercept at RX.

TA : the target angle, from the target, between the target's flight path and the line of sight.

LA : lead angle, from the ship ,between the missile's flight path and the line of sight.

RT : the distance traveled by the target between successives intercepts.

RI : the intercept range, measured from the ship.

As it was stated before, the first missile must be fired so that it will intercept at the maximum range of the missile. The equation will be derived for the additional number of missiles fired after the raid has reached RX, and one missile will be added to this number.

CASE I: target with any heading and no delay time.

An equation giving the number of missiles that can be shot at an incoming threat as a function of the target's closest point of approach is derived below. The geometry involved in this model is shown in Figure IV-3.

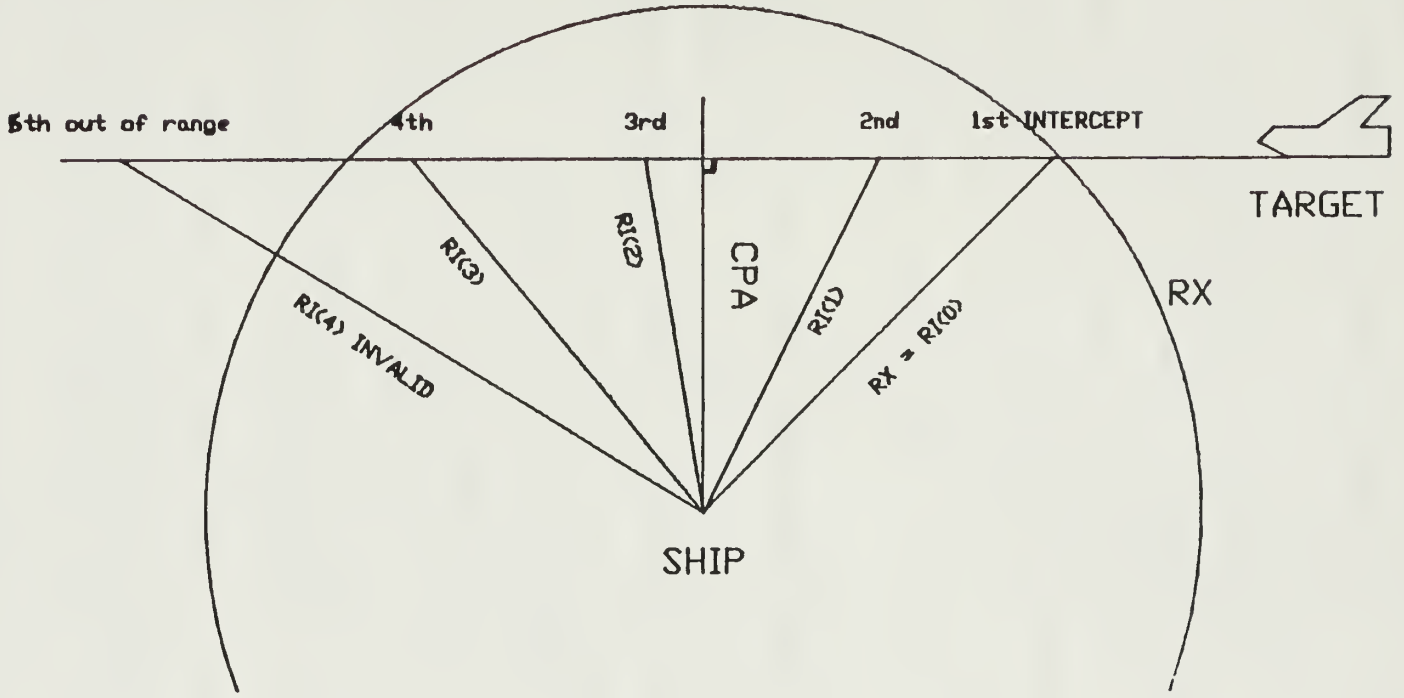

Figure IV-3. Target with a CPA and no delay time.

From this figure it can be seen that the smaller the CPA is, the more missiles that can be fired. The flight path of the target within the missile's envelope increases as the CPA decreases.

The number of missiles that the ship can fire during the time the target is within range is a function of the parameters RX, RN, VM, VT and the missile's flight geometry. Figure IV-4 illustrates how the flight geometry changes with two different laws of navigation (pursuit and lead angle), thus changing the time of flight and the distance to the intercept point.

In the following derivation, the lead angle navigation with a constant line of sight is chosen for three reasons: 1) this course is more efficient in terms of the SAM time of flight before it reaches the target; 2) the equation is easier to develop; 3) it is often used by many missiles.

Figure IV-5 illustrates the problem to be solved in calculating the number of possible shots as a function of the CPA. The initial target angle TA can be defined as

$$TA = \text{Arcsin}(CPA/RX)$$

The lead angle is derived using Figure IV-6. For an intercept course to exist, the components perpendicular to the line of sight of both the target speed (Vta) and the missile speed (Vma) must be equal. Thus

$$Vta = VT * \sin TA$$

$$Vma = VM * \sin LA$$

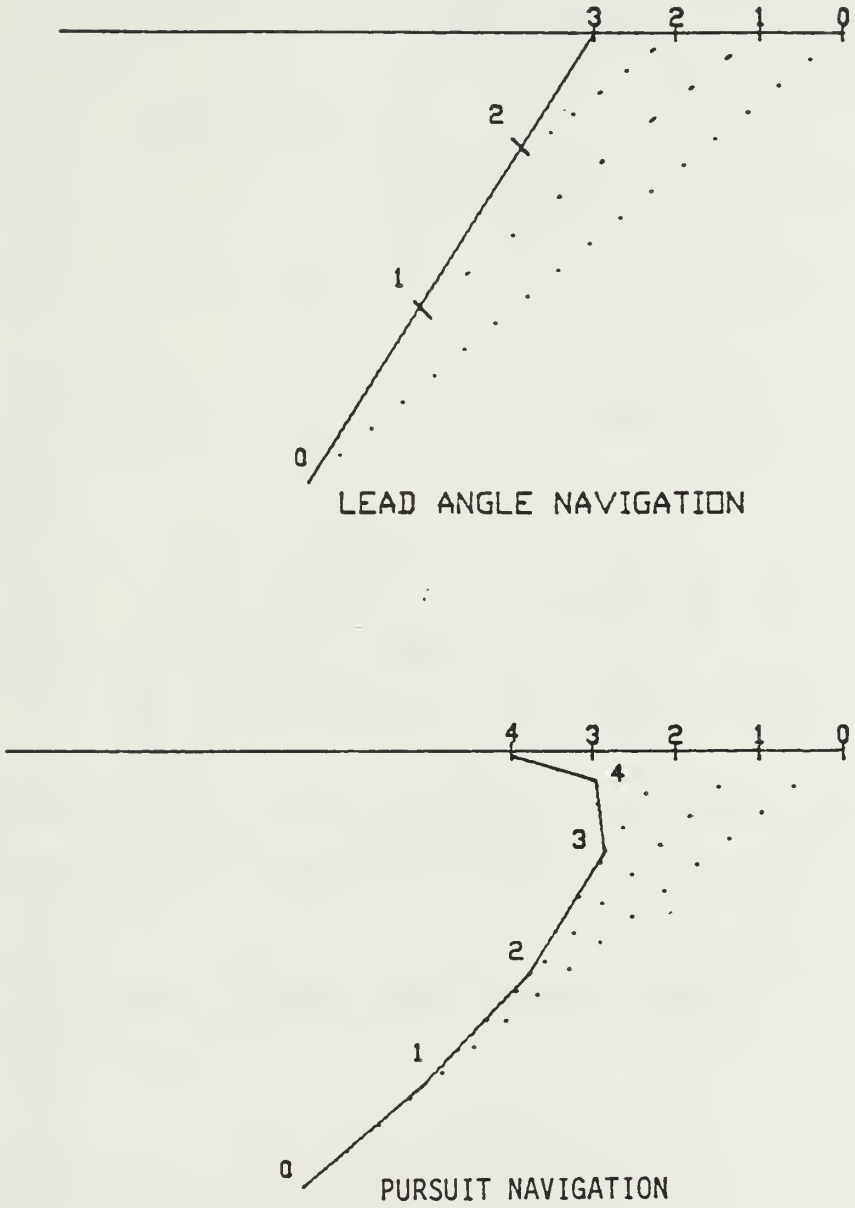

Figure IV-4. Flight geometry changes with two different laws of navigation.

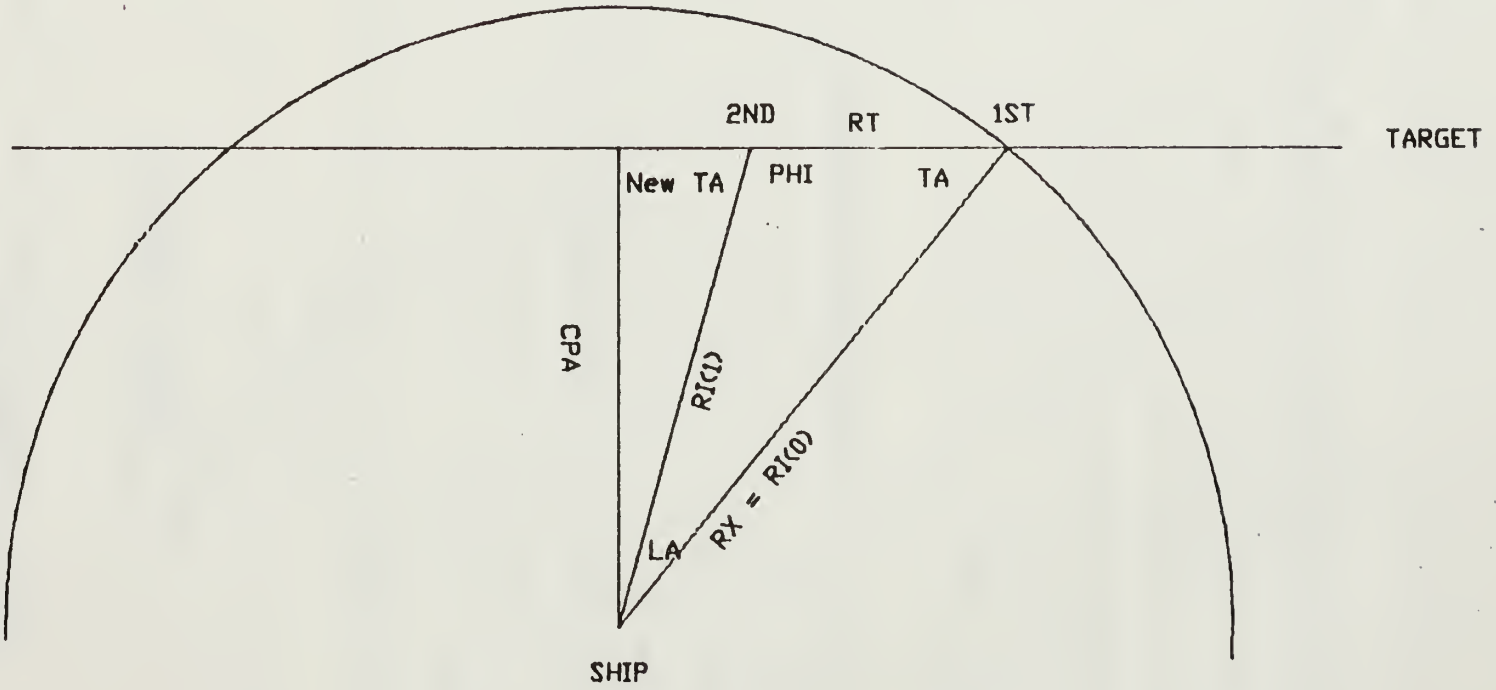

Figure IV-5. Geometry of the engagement with no delay time (Case I).

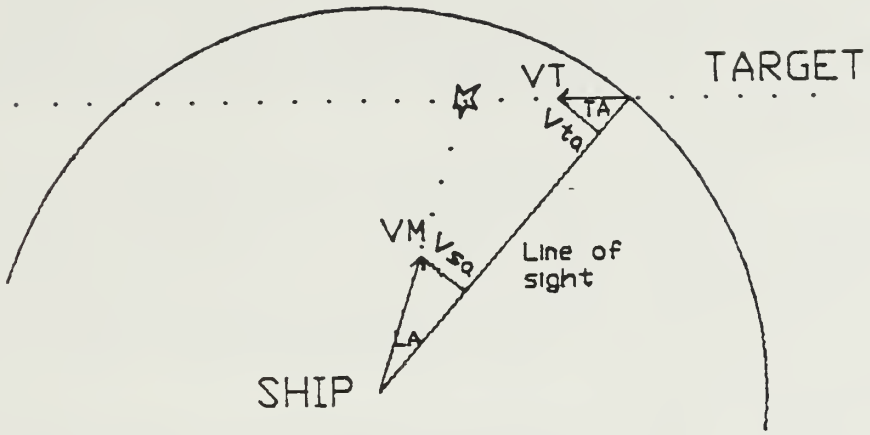

Figure IV-6. Lead angle and target angle.

therefore

$$VT * \sin TA = VM * \sin LA$$

and hence

$$LA = \text{Arcsin}\ [(VT * \sin TA) / VM]$$

Once the lead angle equation is defined, the problem of Figure IV-5 an be solved for RT and RI. Knowing both lead angle and target angle, the third angle (PHI) of the triangle described by RX, RT and RI in Figure IV-5 can be determined by using

$$PHI = PI - (TA + LA)$$

RT and RI can be found by the law of sines

$$\sin PHI / RX = \sin TA / RI = \sin LA / RT$$

$$RI = RX \sin TA / \sin PHI$$

$$RT = RX \sin LA / \sin PHI$$

If a shoot-look-shoot firing doctrine is used, and a zero delay time between shots is assumed, then the intercept range for the previous shot becomes the firing range for the succeeding shot, and the target angle for the succeeding shot can be expressed as

$$TA(n) = Arcsin[CPA / Ri(n-1)]$$

for a closing target, and

$$TA(n) = PI - Arcsin[Rcpa/Ri(n-1)]$$

for an opening target.

Another way to compute this angle regardless of the target's situation is to use the angles of the previous firing

$$TA(n) = PHI(n-1) + LA(n-1)$$

Now the initial lead angle LA can be calculated, and RT(2) and RI(2) calculated, and so on, until the target has flown out of SAM range.

To get the maximum number of missiles, a missile must be fired to intercept exactly at the maximum SAM range. Therefore, each intercept shown in Figure IV-3 is in addition to the initial intercept at maximum range.

The computations of the firing parameters, the intercept range mainly, are iterative. Each firing geometry depends on the previous firing, and so on until the initial intercept which is RX. As long as RI is smaller than RX the iteration will be executed, and the number of missiles that can be fired is incremented. When this condition is not true anymore, the iteration is abondonned and the total number of

missiles to be fired is n plus the first missile for the initial interception.

CASE II: target with any heading and a delay time

For this general case, the target is approaching the ship from any direction on any course as in case 1. However, a delay time T is adopted for observation of the missile at the intercept point to see if another missile is needed or not to kill the target. In the previous case, the intercept point was considered to be the initial firing range for the succeeding missile. In this case, the target will travel a certain distance corresponding to the delay time T before a new engagement geometry will be solved for. This distance is the same for all the shots since the delay time and the target's velocity are fixed. To solve the firepower equation the target angle (TA) after the delay time is required. are needed first. Three new parameters are defined in Figure IV-7:

- NPHI is the angle between the flight path of the target and the previous intercept range RI.
- NL is the angle opposite the distance DIST corresponding to the delay time.

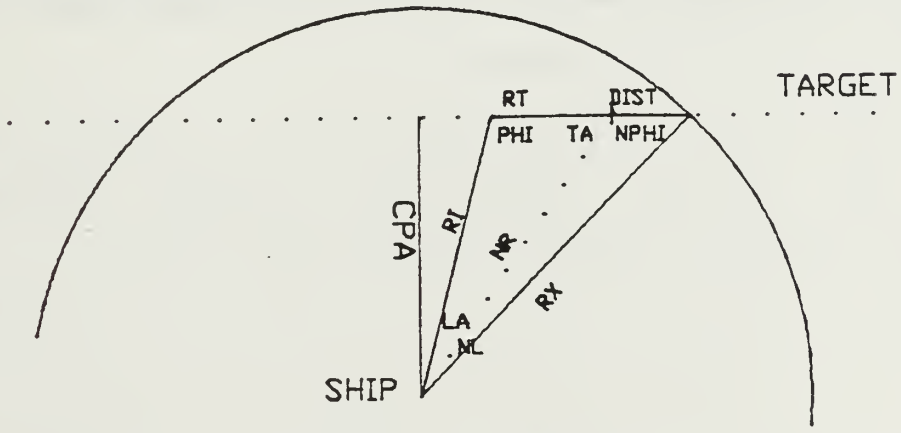

Figure IV-7. Geometry of the engagement with delayed time.

- NR is the new initial firing range used to compute the target angle.

The equation for TA is solved in the same manner as before, but there are more steps in this case. They are:

NPHI = Arcsin[CPA / RI]

DIST = T * VT

NR = SQR {DIST ** 2 + RI ** 2 - 2*DIST*RI*CosNPHI}

NL = Arcsin[DIST * CPA / (NR * RI)]

TA = NPHI + NL

With TA determined, the computation is continued as for the previous case to solve for the RI.

This iteration tests for the next intercept range to see if it is valid to fire another missile or not by comparing RI to RX.

When the intercept of the target is possible but the target itself is far from the ship, it takes a relatively long time for the missile and the target to reach the intercept point. Thus, very few missiles will be expended. But the closer the target is from the ship the shorter is the time needed to reach the intercept point. Thus, many missiles will be expended within a few seconds.

Since many types of missiles have a minimum intercept range, the iteration solving the firepower equation is improved to take the minimum range RN into account.

When the target closes within the minimum range the iteration stops and then restarts immediately with a new intercept range at RN for the departing target. The iteration then continues for the departing target as before.

V. THE INPUT MODULE

This chapter describes the opening screens of ISEAS and the input screens which give the user the choice to change any of the variables.

A. DIFFERENT SCREENS

These screens are all in graphics mode (versus system's text). The graphics library used in ISEAS is HALO, which is considered to be the standard for the Graphics Kernel System (GKS). For detailed information about the use of HALO in ISEAS see Chapter VI. (Graphics).

1. The Opening Screens

When ISEAS is executed, the first screens displayed are the opening screen (Figure V-1) and the explanatory screen. Their function is to inform the user of ISEAS, what does the word ISEAS stands for and its developers. The explanatory screen briefly describes ISEAS, discussing what it does, what is its scope, how to use it, and where to refer to for more information.

2. The Input Screens

As it was described in Chapter III., ISEAS is a program that consists of three different modules; the radar module, the weapons module, and the combat information center module. For versatile use of ISEAS, the program has an input

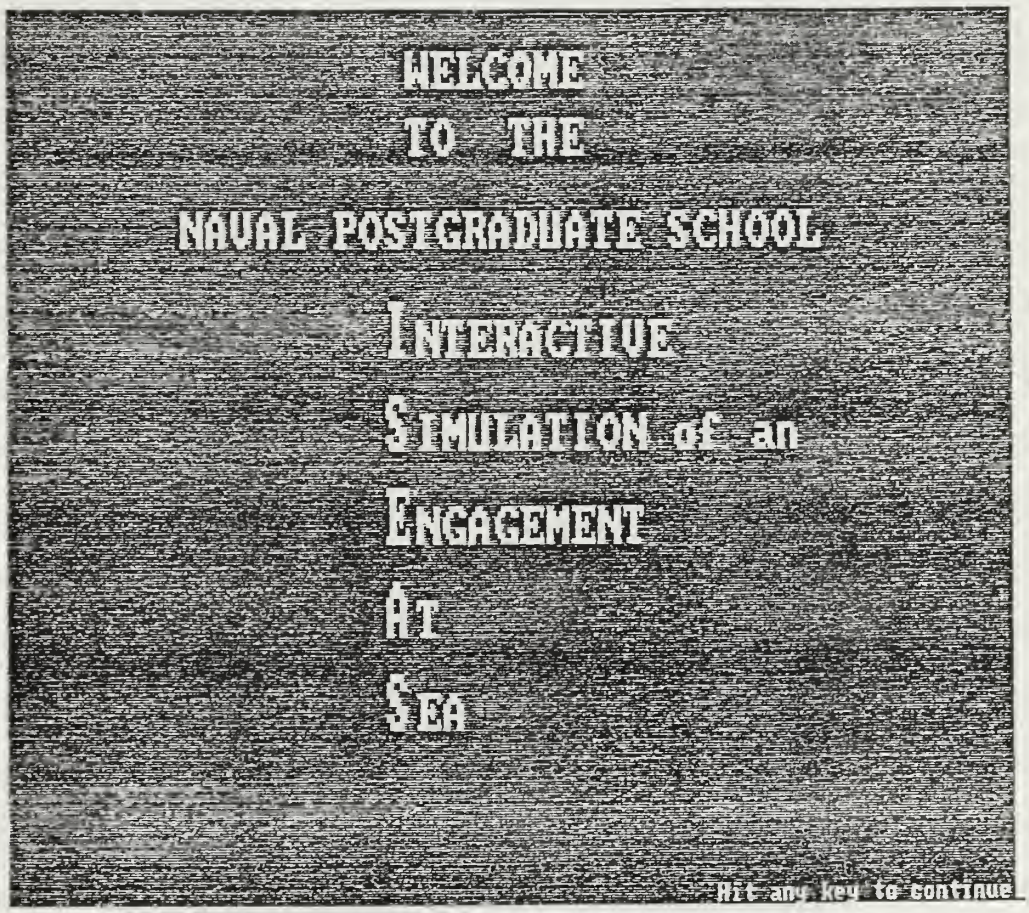

Figure V-1. The Opening Screen.

capability in which the user can change some or all of the variables and judge their influence on the outcome of the simulation.

In ISEAS, every module has an one or more independent input screens such as shown in Figure V-2. Basically, they are all built the same way. First, since they are graphics screen, HALO's environment is initialised and a viewport (i.e window) covering 75% of the screen is set(the other 25% is reserved for the scanning of the values of the variables as explained later). This viewport is filled with a color as a background color and mapped from 0 to 100 on the horizontal, and 0 to 50 on the vertical with the origin (0,0) located at lower left corner of the display monitor. This mapping is necessary so the application can be device independent, and also for the locating of HALO's text cursor. On this viewport, a title for that screen is displayed. Since this is the input module, the title for all the screens is "INPUT MODULE". A subtitle displayed below it informs the user as to which set of variables are actually being input. There are four subtitles and thus four input screens; the search radar, the tracking radar, the weapons, and the combat information center input screens. On each input screen the list of variable names that can be changed is displayed. Each variable name bears a letter (from a to z) as an individual identifier. Note that the values of these variables are not displayed yet. They are

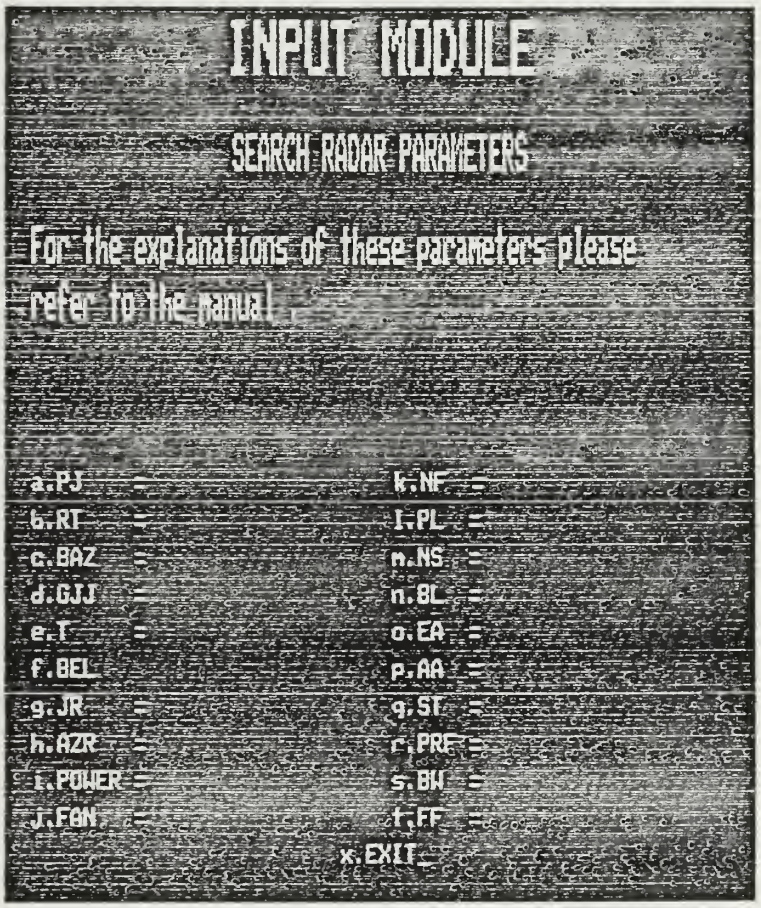

Figure V-2. Search Radar input screen with no values display yet.

stored in disk files which are identified by filenames that match their module name; the extension of these files is "DFL" for default. The program scans the variables values from these files and automatically displays them next to their variables names respectively (Figure V-3). The user is then asked if he wants to change some of the variables. A message at the top of the screen refers the user to the manual (to be prepared) for the meaning and limits of these variables. If the user chooses to change some variables and presses the character 'Y', the program switches to the "input mode". This is a 'while loop' with the condition being i (the input character) equal to 'Y'. A SWITCH instruction branches to the appropriate CASE statement based on the letter entered and identifies the variable name to be changed. The new value is entered at the keyboard. The old value is erased then (Figure V-4), and the value newly entered is displayed. As long as i is equal to 'Y' the user is in the input mode which allows him to change any variable as many times as he whiches and in any order he wants. He can leave the input mode by pressing the character 'x' which results in the branching to the CASE 'x' where i is set equal to 'N'. The condition for the while loop is not true anymore and the program exits the input mode. The display of the values on these input screens was not easy to create. HALO has no capabilities of inputing or outputing floating numbers from the console or the keyboard and no text can be sent directly from the keyboard. It displays text (not numbers

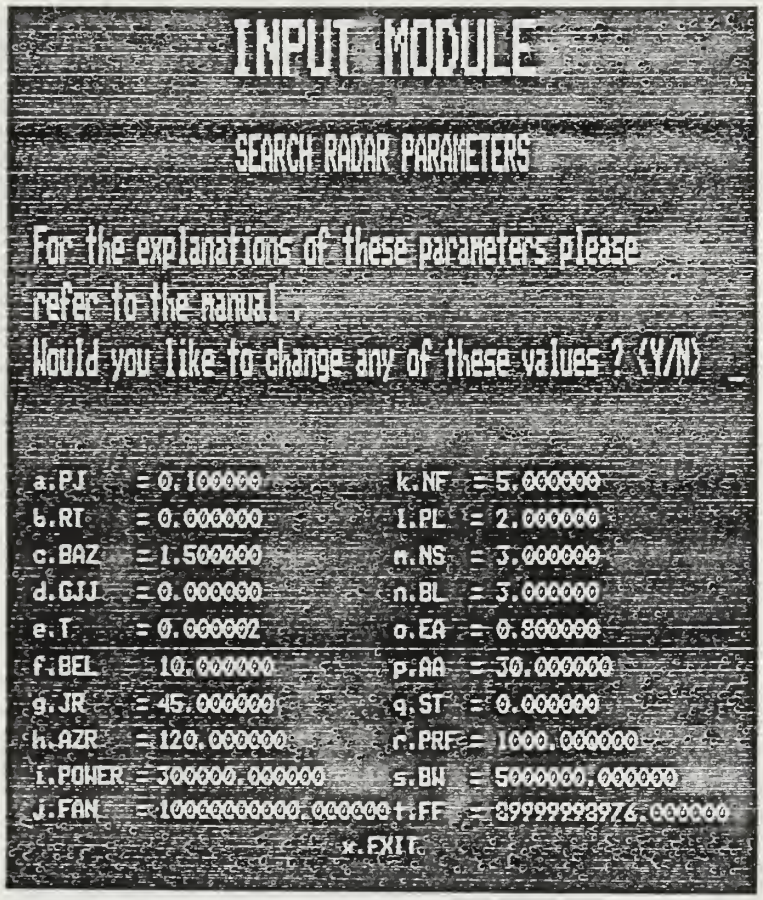

Figure V-3. Search Radar input screen with the default values displayed.

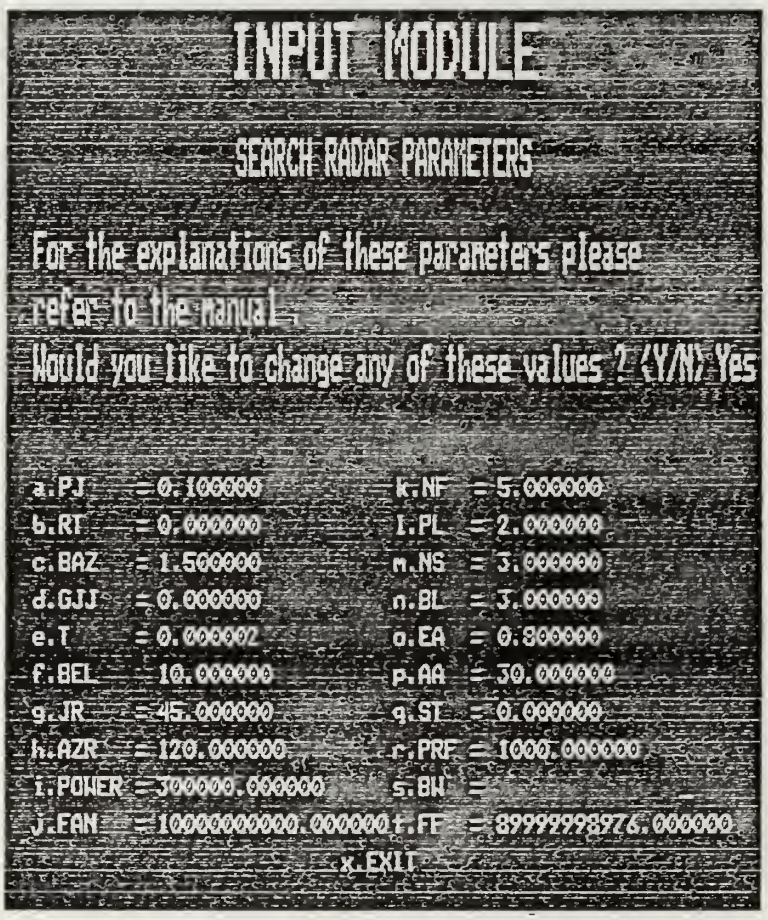

Figure V-4. Input screen with a value [s.BW] to be changed.

such intergers or float values) that was stored in a string. Thus all the variables values are converted from integer or float numbers to character text! This was done with the help of the Lattice C compiler, except that scanned variables are echoed on the display monitor, which destroys the graphics on the viewport! The method used to fix this inconvenience was the allocation of 25% of the total area of the display for text input and number input. The rest was allocated to the graphics with the viewport defining the limits between the two modes.

When the user exits the input mode he is asked if he wishes to save his set of data in a disk file (for future use, reference or comparison) (Figure V-5). In the affirmative, he is asked to enter the filename and extension of his file. Since it is a disk file the filename can be no longer than eight characters, and the extension no longer than three characters. After that, the program opens a disk file by that name and stores the data in it. This last action ends the current input screen, and the user is prompted to hit any key to continue. A new input screen is displayed with the corresponding title and variables list, and so on, until all the input screens are displayed.

ISEAS consists of two separate programs. The first program, the input program, is used to change variables and to create data files. The second program is the actual simulation. The later asks for a data file to load. The user

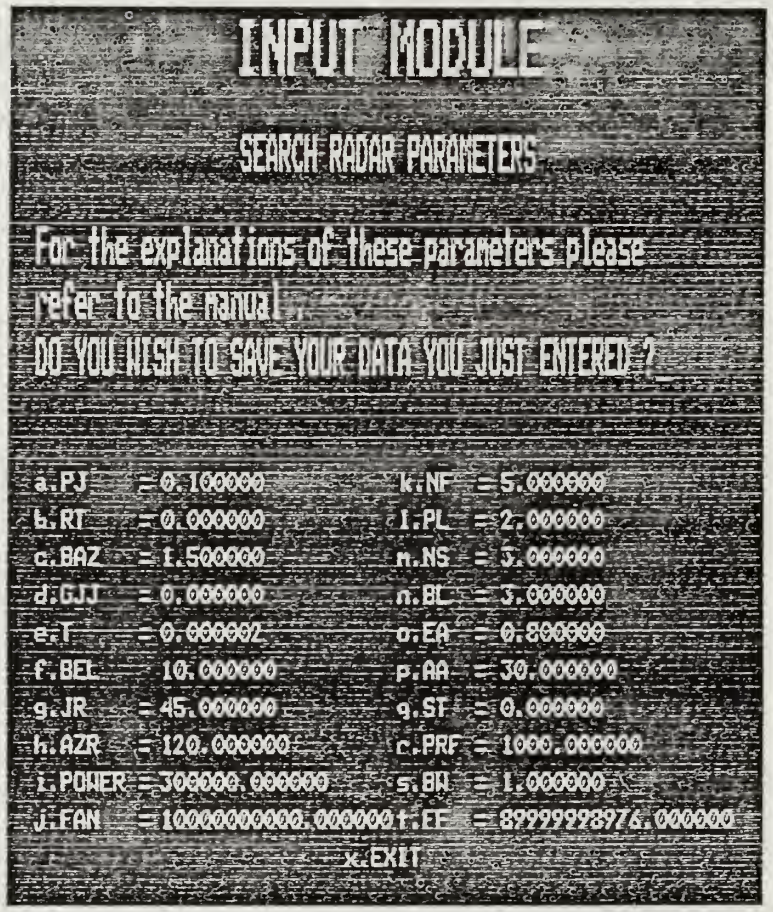

Figure V-5. An input screen with the option to save the data.

will either choose the default files or his own, which was created by the first program. The default files are provided with the program. But if they are not available for a reason of a loss or other reason, they can be generated easily. The variables will be zeroed out because of their nonavailability (Figure V-6) but they can be changed to new default values and stored in the default files. The same procedure can be followed if the default files need to be changed.

B. WHAT ARE THE OPTIONS?

The set of variables used in ISEAS is divided into three parts according to the three different modules. In the following, these parts are described as to how many screens each part is using and the list of variables displayed on that screen with the definition of each variable.

1. RADAR INPUT

This module has two input screens; the search radar screen and the tracking radar screen.

a. Search radar screen (Figure V-3)

This screen has twenty variables, their default values are stored in the file called "SEARCH.DFL". Their definitions, units and default values are:

a.PJ : jammer power, 0.1 Watts.

b.RT : recovery time, 0.5E-6 seconds.

c.BAZ : azimuth beam width, 1.5 degrees.

d.GJJ : jammer antenna gain, 0.0.

e.T : pulse width, 2E-6 seconds.

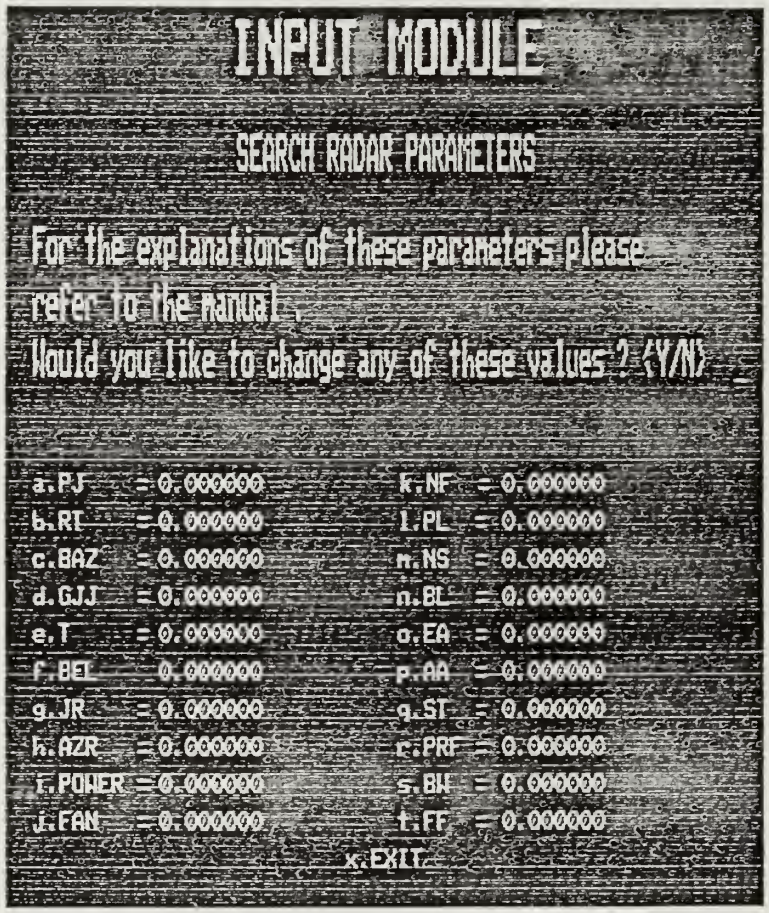

Figure V-6. An input screen with the variables "zeroed out"

f.BEL : elevation beam width, 10.0 degrees.

g.JR : jammer to radar distance, 45.0 meters.

h.AZR : antenna scan rate, 120.0.

i.POWER: transmitter power, 3.0E5 Watts.

j.FAN : false alarm number, 1.0E10 ?

k.NF : noise figure, 5.0 ?

l.PL : plumbing loss, 2.0 (dB ?)

m.NS : number of scans, 3.

n.BL : beam loss, 3.0 (dB ?).

o.EA : effective area, 0.8 square meters.

p.AA : antenna altitude, 30 meters.

q.ST : system temperature, 290.0 degrees Kelvin.

r.PRF : pulse repetition frequency, 1000.0 Hertz.

s.BW : receiver bandwith, 5.0E6 Hertz.

t.FF : transmitting frequency, 9.0E9 Hertz.

x.EXIT : quits the input mode.

b. Tracking radar screen (Figure V-7)

This screen has twelve variables. The variables names are similar to those of the search radar. However, to differentiate them from the previous ones they all have the letter T at the end (for Tracker). The values are different, and some variables found in the search radar screen don't exist in the tracking radar screen. These variables are stored in a file named "TRACK.DFL". For more details refer to Ref. 1.

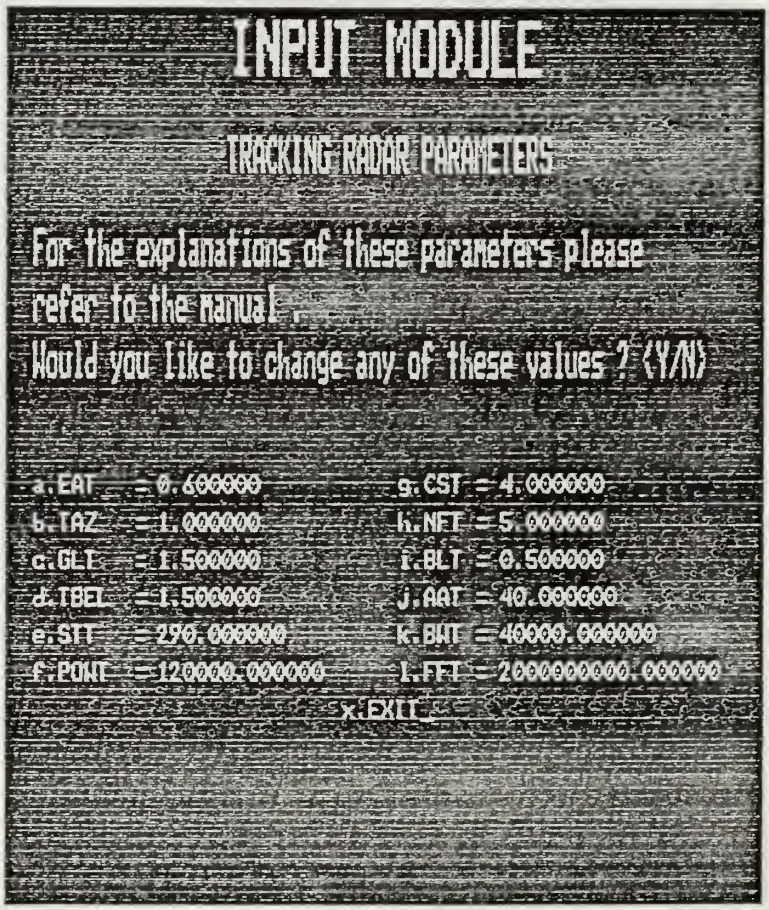

Figure V-7. Tracking Radar Input Screen.

2. CIC INPUT

This module has one input screen. Its variables are of the ship's situation, and the CIC. The variable values are stored in the file called "CIC.DFL".

a.CS: ship's course, 0.0 degrees.

b.VS: ship's speed, 25.0 Km/h .

c.T : time increment for the simulation, 0.1 seconds.

d.LT: look time for the look-shoot-look policy, 4 seconds.

e.TK: treshold for deciding a kill has occured, 99.5% .

x.EXIT.

3. WEAPONS INPUT

This module was not available at the time of writing this thesis.

VI. GRAPHICS

As was discussed in the Introduction, the graphics represents a large part of this thesis. One of the best ways to take full advantage of the stream of information flowing in the CIC is to display it in the form of graphics on tactical display CRTs.

In ISEAS, the critical information is gathered from other modules, processed and displayed graphically on the screen, simulating what is actually happening or what might happen in the near future. Thus, the user is saved a big step of imagining in his mind, the scenario of the engagement.

A. HALO

The graphics portion of ISEAS is written in the C language with the graphics library HALO. The later follows the Graphics Kernel System (GKS). HALO, version 2.30, and lattice C, version 3.00G, are the actual library and compiler used.

HALO is a library of subroutines offering the programmer the possibility of adding graphics to his application program. In the following paragraphs, some concepts used in ISEAS are explained permitting the user to understand how the simulation is conducted and also permitting an improvement of ISEAS.

HALO has first to be initiated.. During this initiation the interfacing devices used have to be defined, especially the type of monitor used. Once the graphics mode is initiated

and the appropriate device driver is loaded, graphics are then possible to draw. The actual version of ISEAS is designed to run with either the IBM Color Graphics Adaptor (CGA) or the IBM Enhanced Graphics Aaptor (EGA). Any modification done so that ISEAS can run on another type of monitor can be easily done in the source code.

ISEAS is fairly device independent. To achieve this, the world coordinates are used throughout the program. The world coordinates (WC) are a set of coordinates, user defined, that map the screen to the range of coordinates chosen by the programmer. A point is located with these coordinates regardless of the resolution of the monitor used. In ISEAS, the world coordinates range from zero to one hundred on each axis for all of the graphics screens. The viewport (or window) is the part of the monitor dedicated to graphics. The whole surface of the monitor could be dedicated to one graphics screen, the limits of the viewport are then (0.0,1.0) on each axis (i.e., the whole length of each side is used as limit for the viewport). Some screens in ISEAS use the whole display screen for one graphics screen. But mostly two viewports share the same display screen. The reasons for this partition are:

a. two different viewports are needed, one for the graphics of the simulation and the other for the display of the information in text form.

b. the surface of the monitor is not square. Thus, anytime a location is addressed, its coordinates have to be scaled to the aspect ratio. Otherwise, the range on the horizontal axis will be larger than the vertical axis. This will not fit the circular view given by the radar.

Because of this, a square viewport used by the graphics is located to the left and it ranges from 0.0 to 0.6325 on the horizontal axis (x-axis) and 0.0 to 1.0 on the vertical axis (y-axis), (0.0,0.0) being the lower left corner.

Non-destructive graphics are wanted most of the time. This is when the background or whatever was previously displayed is restored once the present image is 'erased'. Erasing an image is merely rewriting it twice. The second time it is written, the image is XORED with the first one and the background is restored. This is achieved by turning the XOR mode on. Animation is also realized by the same procedure. A figure is drawn once, so that it appears on the screen, then it is drawn a second time so that it is 'erased', then the coordinates of the figure are shifted toward the direction of the motion and redrawn and so on. One problem arises when using this mode with a colored background; the colors change and some of them don't show at all. The background used in ISEAS is light blue and the colors used are carefully chosen so that the user sees the screen (and the simulation and data) easily.

B. THE SIMULATION SCREEN

The graphics are used mainly to show what is physically happening during the engagement. Also, possible future engagement geometries, intercept points and range limitations are shown on the graphics screen. The preengagement phase and

the engagement itself can be followed on the different types of screens. The first one is a radar-type display, where a circular view is adopted with the ship permanently located in the center. The second screen is a TV-type display (not discussed here), where the engagement is followed on the three different planes. Both of them have a viewport reserved for text, messages and data displaying. In the following these types of graphics screen are explained in details.

The Radar-type display is the first screen displayed (Figure VI-1), and it is used for the general information about the tactical situation and the ship's situation. It has two viewports, a square one on the left that simulates a radar scope display, a second one to the right where messages and text information are displayed periodically.

On the first viewport, the ship is located in the center. A pair of orthogonal axes are drawn with marks along them. Then four concentric circles are displayed. The outer one is fixed for all the simulations, and it represents the maximum detection range of the search radar used, it is labeled "DR". The other three circles represent the ranges for the long range, medium range and short range missiles on board. No other information is displayed before the detection of a target.

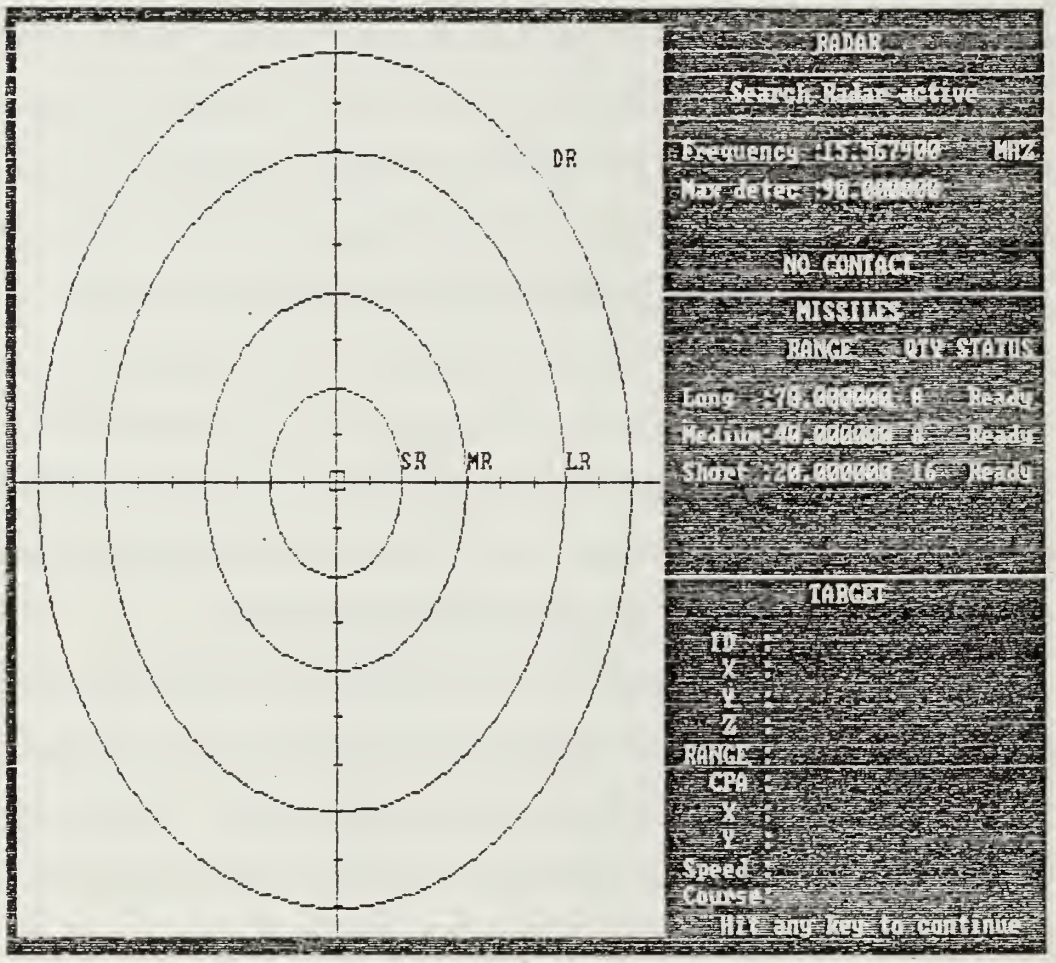

Figure VI-1. Tactical screen with no contact made yet.

The second viewport (Figure VI-1) is divided into three windows showing information about the radar used, the missiles and the target. The first window displays information about the radar; the search radar or tracking radar used, then the operating frequency, then the maximum detection range, and finally a message about the contact. At this stage, the message "NO CONTACT" is displayed. The second window contains information about the missiles. Their ranges, available number for each one and status (ready/not ready) are made known to the user. The third window, and the most important, gives informations about the target once it is detected. Before its detection, only the titles of the parameters are displayed with no data available. These parameters are the identity of the contact (ID) designating it as friend, enemy or unknown. Then the present position of the contact from the ship is displayed. The CPA and its position are also displayed. Finally, the speed and course of the contact are shown. The current target is non-maneuvering, thus the only information to be updated constantly in this window is the position and the range. When a contact is made, and that happens only when the target is within the detection range, its position and course are plotted instantaneously, then the CPA is drawn showing its location relative to the ship. On the 'text' viewport first window, the message "NO CONTACT" is replaced by a blinking highlighted message "WARNING! CONTACT". The third window displays all the information available about

the contact and updates its position while it is plotted on the 'graphics' viewport simultaneously. The target keeps flying, and the user takes action. The choices are discussed later, but if the target leaves the detection envelope, the updating of the position and the warning signal stop. A new message "CONTACT LOST!" is displayed, and the search continues for a new contact.

During the simulation of the target's flight, the CIC module solves the firepower equation for the geometrically possible intercept points for the three different range missiles. Then these intercept points are displayed on the screen. The combination of the circles for the three ranges and these intercept points gives a complete idea to the user of what are his capabilities regarding that specific target. The user has the possibility to fire the missile of his choice at a time he judges appropriate. His judgement will be based on what he sees on the screen regarding the present tactical situation and his fire power capabilities. And that is the main idea behind using the graphics in the CIC.

VII. SUMMARY

A computer program simulating an anti-air operation conducted from the C.I.C. of a ship was written in the C language to run on an MSDOS personal computer. This program simulates teh main functions of a C.I.C. and is incorporated into the NPS interactive simulatiron of an engagement at sea (ISEAS) with a radar module and a weapons module developed by others. The contribution of this thesis to ISEAS is weapons direction, decision-making and graphics display of the tactical information in a useful form.

The present version of ISEAS is the product of a first iteration which was dedicated mostly to define the frame work and scope. It is intended to demonstrate basic features found in any CIC, and encounter between a ship and an attacking aircraft.

The author hopes that more students will take this task over and improve the present version and enhance it to a higher level with more capabilities. It is a task that can be carried to a high level of expertise.

APPENDIX A

```
#include"stdio.h"
#include"stdlib.h"

float x1 = 0.0 , y1 = 0.0  ;
float x2 = 100.0,y2 = 50.0 ;
int black  = 0  ;
int blue   = 9  ;
int green  = 2  ;
int red    = 12 ;
int yellow = 14 ;
int white  = 15 ;
int purple = 5  ;
int liblue = 3  ;
float PRF, AA, T, RT, EA, FF, PL, BL, BW, ST, NF,
      POWER, AZR, NS, BAZ, BEL, FAN, PJ, GJJ, JR ;
float AAT,EAT,BWT,CST,POWT,FFT,GLT,BLT,STT,NFT,TAZ,TBEL ;
float cx , cy ;
int h , w , p , m ;
int i ;
int pixels ;
int switsh = 1 ;
float vx1=0.25,vy1=0.0,vx2=1.0,vy2=1.0 ;
char s[20] ;
int border = -1 , back = -1 ;


main()
 {
  int mode ;
  setdev("HALOIBME.DEV") ;
  mode = 4 ;
  initgraphics(&mode) ;
  setworld(&x1,&y1,&x2,&y2) ;
  Opening() ;
  intro()   ;
  input1()  ;
  change1() ;
  input2()  ;
  change2() ;
  closegraphics() ;
 }
```

```
Opening()
 {
  float cx , cy ;
  int h , w , p , m ;
  int i ;
  int pixels ;

  pixels = 13 ;
  setlnwidth(&pixels) ;
  setcolor(&red)  ;
  box(&x1,&y1,&x2,&y2);
  cx = 50.0  ;
  cy = 49.0  ;
  movabs(&cx,&cy)     ;
  flood(&liblue)      ;
  h = 2 ;
  w = 2 ;
  p = 0 ;
  m = 0 ;
  settext(&h,&w,&p,&m) ;
  settextclr(&blue,&black). ;
  cx = 40.0 ;
  cy = 44.5 ;
  movtcurabs(&cx,&cy) ;
  text("WELCOME")  ;
  cx = 40.0 ;
  cy = 41.0 ;
  movtcurabs(&cx,&cy) ;
  text("TO  THE")  ;
  cx = 18.0 ;
  cy = 36.0 ;
  settextclr(&black,&black) ;
  movtcurabs(&cx,&cy) ;
  text("NAVAL POSTGRADUATE SCHOOL") ;
  h = 3 ;
  settext(&h,&w,&p,&m) ;
  settextclr(&red,&black) ;
  cx = 38.0 ;
  cy = 30.0 ;
  movtcurabs(&cx,&cy) ;
  text("I") ;
  cy = 25.0  ;
  movtcurabs(&cx,&cy) ;
  text("S") ;
  cy = 20.0 ;
  movtcurabs(&cx,&cy) ;
  text("E") ;
  cy = 15.0 ;
 movtcurabs(&cx,&cy) ;
 text("A") ;
 cy = 10.0 ;
```

```
movtcurabs(&cx,&cy) ;
text("S") ;
h = 2 ;
settext(&h,&w,&p,&m) ;
settextclr(&yellow,&black) ;
cx = 41.0 ;
cy = 30.2 ;
movtcurabs(&cx,&cy) ;
text("NTERACTIVE") ;
cy = 25.2 ;
movtcurabs(&cx,&cy) ;
text("IMULATION of an") ;
cy = 20.2  ;
movtcurabs(&cx,&cy) ;
text("NGAGEMENT") ;
cy = 15.2 ;
movtcurabs(&cx,&cy) ;
text("T") ;
cy = 10.2 ;
movtcurabs(&cx,&cy) ;
text("EA") ;

h = 1 ;
w = 1 ;
settext(&h,&w,&p,&m) ;
settextclr(&black,&black) ;
cx = 70.0  ;
cy =  1.0  ;
movtcurabs(&cx,&cy) ;
text("Hit any key to continue") ;
i = getch()  ;
setcolor(&black) ;
clr() ;
return   ;
}
```

```
intro()
{
 float cx , cy ;
 int h , w , p , m ;
 int i ;
 int pixels ;

 pixels = 13 ;
 setlnwidth(&pixels) ;
 setcolor(&red)  ;
 box(&x1,&y1,&x2,&y2):
 cx = 50.0  ;
 cy = 49.0  ;
 movabs(&cx,&cy)     ;
 flood(&liblue)      ;
 h = 3 ; w = 2 ;
 p = 0 ; m = 0 ;
 settext(&h,&w,&p,&m) ;
 settextclr(&blue,&black) ;
 cx = 34.0 ; cy = 42.0 ;
 movtcurabs(&cx,&cy) ;
 text("INTRODUCTION");
 h = 1 ; w = 1 ;
 settext(&h,&w,&p,&m) ;
 settextclr(&purple,&black) ;
 cx = 14.0 ; cy = 36.0 ;
 movtcurabs(&cx,&cy) ;
 text("This program simulates a  Combat Information Center  in
action") ;
 cx = 10.0 ; cy = 34.0 ;
 movtcurabs(&cx,&cy)  ;
 text("in  an  Anti-Air  operation .  A target  flying  at  a
constant velocity") ;
 cy = 32.0 ;
 movtcurabs(&cx,&cy) ;
 text("and at a fixed altitude is generated.  The ship has for
mission the") ;
 cy = 30.0 ;
 movtcurabs(&cx,&cy) ;
 text("countering  of  this  target  and  ultimately  its
destruction .") ;
 cy = 22.0 ;
 movtcurabs(&cx,&cy) ;
 text("   THIS PAGE WILL BE RESERVED FOR A SHORT  INTRODUCTION
TO THE PROGRAM");

 h = 1 ;
 w = 1 ;
 settext(&h,&w,&p,&m) ;
 settextclr(&black,&black) ;
 cx = 70.0  ;
```

```
 cy =  1.0  ;
 movtcurabs(&cx,&cy) ;
 text("Hit any key to continue") ;

 i = getch() ;
 setcolor(&black) ;
 clr() ;
 return ;
}
```

```
input1()
{
 FILE *pointer1 ;

 setviewport(&vx1,&vy1,&vx2,&vy2,&border,&back) ;
 pixels = 13 ;
 setlnwidth(&pixels) ;
 setcolor(&red)  ;
 box(&x1,&y1,&x2,&y2);
 cx = 50.0  ;
 cy = 49.0  ;
movabs(&cx,&cy)     ;
 flood(&liblue)      ;
 setxor(&switsh)     ;
 h = 3 ; w = 2 ;
 p = 0 ; m = 0 ;
 settext(&h,&w,&p,&m) ;
 settextclr(&green,&black) ;
 cx = 30.5 ; cy = 45.0 ;
 movtcurabs(&cx,&cy) ;
 text("INPUT MODULE") ;
 settextclr(&red,&black) ;
 h = 2 ; w = 1 ;
 settext(&h,&w,&p,&m) ;
 cx = 31.0 ; cy = 40.0 ;
 movtcurabs(&cx,&cy) ;
 text("SEARCH RADAR PARAMETERS") ;
 settextclr(&purple,&black) ;
 cx =  5.0 ; cy = 35.0 ;
 movtcurabs(&cx,&cy) ;
 text("For the explanations of these parameters please ");
 cy = 32.0 ;
 movtcurabs(&cx,&cy) ;
 text("refer to the manual . ") ;

 h = 1 ;
 settext(&h,&w,&p,&m) ;
 cx =  5.0 ; cy = 23.0 ;
 movtcurabs(&cx,&cy) ;
 text("a.PJ    =                    k.NF = ") ;
 cy = 21.0  ;
 movtcurabs(&cx,&cy) ;
 text("b.RT    =                    l.PL  = ") ;
 cy = 19.0  ;
 movtcurabs(&cx,&cy) ;
 text("c.BAZ   =                    m.NS  = ") ;
 cy = 17.0  ;
 movtcurabs(&cx,&cy) ;
 text("d.GJJ   =                    n.BL  = ") ;
 cy = 15.0  ;
 movtcurabs(&cx,&cy) ;
```

```
 text("e.T      =                    o.EA   = ") ;
 cy = 13.0    ;
 movtcurabs(&cx,&cy) ;
 text("f.BEL    =                    p.AA   = ") ;
 cy = 11.0   ;
 movtcurabs(&cx,&cy)   ;
 text("g.JR     =                    q.ST   = ") ;
 cy = 9.0 ;
 movtcurabs(&cx,&cy) ;
 text("h.AZR    =                    r.PRF  = ") ;
 cy = 7.0 ;
 movtcurabs(&cx,&cy) ;
 text("i.POWER =                     s.BW   = ") ;
 cy = 5.0 ;
 movtcurabs(&cx,&cy) ;
 text("j.FAN    =                    t.FF   = ") ;
 cx = 45.0 ; cy = 3.0 ;
 movtcurabs(&cx,&cy) ;
 text("x.EXIT");
 pointer1 = fopen("search.df1","r") ;
 fscanf(pointer1,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f
 %f",
&PRF,&AA,&T,&RT,&EA,&FF,&PL,&BL,&BW,&ST,&NF,&POWER,&AZR,&NS,&BAZ,&BE
 &FAN,&PJ,&GJJ,&JR ) ;
 sprintf(s,"%f", PJ) ;
 cx = 21.0 ; cy = 23.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , NF ) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , RT ) ;
 cx = 21.0 ; cy = 21.0 ;
 movtcurabs(&cx,&cy) ;
 text(s)  ;
 sprintf(s , "%f" , PL ) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , BAZ ) ;
 cx = 21.0 ; cy = 19.0 ;
 movtcurabs(&cx,&cy) ;
 text(s)  ;
 sprintf(s , "%f" , NS ) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , GJJ) ;
 cx = 21.0 ; cy = 17.0 ;
```

```
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , BL ) ;
cx = 65.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , T  ) ;
cx = 21.0 ; cy = 15.0 ;
movtcurabs(&cx,&cy) ;
text(s)  ;
sprintf(s , "%f" , EA );
cx = 65.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , BEL ) ;
cx = 21.0 ; cy = 13.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , AA ) ;
cx = 65.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , JR ) ;
cx = 21.0 ; cy = 11.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , ST ) ;
cx = 65.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , AZR ) ;
cx = 21.0 ; cy = 9.0 ;
movtcurabs(&cx,&cy) ;
text(s)  ;
sprintf(s , "%f" , PRF ) ;
cx = 65.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , POWER ) ;
cx = 21.0 ; cy = 7.0 ;
movtcurabs(&cx,&cy) ;
text(s) ;
sprintf(s , "%f" , BW ) ;
cx = 65.0 ;
movtcurabs(&cx,&cy) ;
text(s)  ;
sprintf(s ,"%f" , FAN ) ;
cx = 21.0 ; cy = 5.0 ;
movtcurabs(&cx,&cy) ;
text(s)  ;
sprintf(s , "%f" , FF ) ;
```

```
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 return    ;
}
```

```
change1()
{
 int  c ;
 FILE *pointer2 ;
 static char filename[13] ;

 h = 2 ; w = 1 ;
 p = 0 ; m = 0 ;
 settext(&h,&w,&p,&m) ;
 cx = 5.0 ; cy = 29.0 ;
 movtcurabs(&cx,&cy) ;
 text("Would  you like to change any of these values  ?  <Y/N>
") ;
 cx = 93.0 ; cy = 29.0 ;
 movtcurabs(&cx,&cy) ;
 i = getch() ;
 if ( i == 'n' )
  text("No")  ;
 if ( i == 'y' )
 {
  text("Yes") ;
  while ( i  == 'y' )
  {
   h = 1 ;
   settext(&h,&w,&p,&m) ;
   c = getch() ;
   switch(c)
   {
    case 'a' :
    cx = 21.0 ; cy = 23.0 ;
    movtcurabs(&cx,&cy) ;
    sprintf(s ,"%f" , PJ ) ;
    text(s) ;
    movtcurabs(&cx,&cy) ;
    scanf("%f" , &PJ ) ;
    sprintf(s,"%f", PJ ) ;
    text(s)  ;
    break ;
    case 'b' :
    cx = 21.0 ; cy = 21.0 ;
    movtcurabs(&cx,&cy) ;
    sprintf(s ,"%f" , RT ) ;
    text(s) ;
    movtcurabs(&cx,&cy) ;
    scanf("%f" , &RT ) ;
    sprintf(s,"%f", RT ) ;
    text(s)  ;
    break ;
    case 'c' :
    cx = 21.0 ; cy = 19.0 ;
    movtcurabs(&cx,&cy) ;
```

```
sprintf(s ,"%f" , BAZ ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &BAZ ) ;
sprintf(s,"%f", BAZ) ;
text(s)  ;
break ;
case 'd' :
cx = 21.0 ; cy = 17.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , GJJ ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &GJJ ) ;
sprintf(s,"%f", GJJ) ;
text(s)  ;
break ;
case 'e' :
cx = 21.0 ; cy = 15.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , T ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &T ) ;
sprintf(s,"%f", T ) ;
text(s)  ;
break ;
case 'f' :
cx = 21.0 ; cy = 13.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , BEL ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &BEL ) ;
sprintf(s,"%f", BEL ) ;
text(s)  ;
break ;
case 'g' :
cx = 21.0 ; cy = 11.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , JR  ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &JR  ) ;
sprintf(s,"%f", JR  ) ;
text(s)  ;
break ;
case 'h' :
cx = 21.0 ; cy = 9.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , AZR ) ;
```

```
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &AZR ) ;
sprintf(s,"%f", AZR ) ;
text(s)  ;
break ;
case 'i' :
cx = 21.0 ; cy = 7.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , POWER ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &POWER ) ;
sprintf(s,"%f", POWER ) ;
text(s)  ;
break ;
case 'j' :
cx = 21.0 ; cy = 5.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , FAN ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &FAN ) ;
sprintf(s,"%f", FAN ) ;
text(s)  ;
break ;
case 'k' :
cx = 65.0 ; cy = 23.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , NF ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &NF ) ;
sprintf(s,"%f", NF ) ;
text(s)  ;
break ;
case 'l' :
cx = 65.0 ; cy = 21.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , PL ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &PL ) ;
sprintf(s,"%f", PL ) ;
text(s)  ;
break ;
case 'm' :
cx = 65.0 ; cy = 19.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , NS ) ;
text(s) ;
```

```
movtcurabs(&cx,&cy) ;
scanf("%f" , &NS ) ;
sprintf(s,"%f", NS ) ;
text(s)  ;
break ;
case 'n' :
cx = 65.0 ; cy = 17.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , BL ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &BL ) ;
sprintf(s,"%f", BL ) ;
text(s)  ;
break ;
case 'o' :
cx = 65.0 ; cy = 15.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , EA ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &EA) ;
sprintf(s,"%f", EA ) ;
text(s)  ;
break ;
case 'p' :
cx = 65.0 ; cy = 13.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , AA ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &AA ) ;
sprintf(s,"%f", AA ) ;
text(s)  ;
break ;
case 'q' :
cx = 65.0 ; cy = 11.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , ST ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f", &ST ) ;
sprintf(s,"%f", ST ) ;
text(s)  ;
break ;
case 'r' :
cx = 65.0 ; cy = 9.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , PRF) ;
text(s) ;
movtcurabs(&cx,&cy) ;
```

```
    scanf("%f" , &PRF) ;
    sprintf(s,"%f", PRF) ;
    text(s)  ;
    break ;
    case 's' :
    cx = 65.0 ; cy = 7.0 ;
    movtcurabs(&cx,&cy) ;
    sprintf(s ,"%f" , BW ) ;
    text(s) ;
    movtcurabs(&cx,&cy) ;
    scanf("%f" , &BW ) ;
    sprintf(s,"%f", BW ) ;
    text(s)  ;
    break ;
    case 't' :
    cx = 65.0 ; cy = 5.0 ;
    movtcurabs(&cx,&cy) ;
    sprintf(s ,"%f" , FF ) ;
    text(s) ;
    movtcurabs(&cx,&cy) ;
    scanf("%f" , &FF ) ;
    sprintf(s,"%f", FF ) ;
    text(s)  ;
    break ;
    case 'x' :
    h = 2 ; w = 1 ;
    settext(&h,&w,&p,&m) ;
    cx = 93.0 ; cy = 29.0 ;
    movtcurabs(&cx,&cy) ;
    text("Yes") ;
    cx = 5.0 ; cy = 29.0 ;
    movtcurabs(&cx,&cy) ;
text("Would you like to change any of these values ? <Y/N>  ")
;
    movtcurabs(&cx,&cy) ;
    text("DO YOU WISH TO SAVE YOUR DATA YOU JUST ENTERED ?") ;
    i = getch() ;
    if (i == 'y')
    {
     movtcurabs(&cx,&cy) ;
     text("DO YOU WISH TO SAVE YOUR DATA YOU JUST ENTERED ?") ;
     movtcurabs(&cx,&cy) ;
     text("WHAT IS YOUR DATA FILENAME ? <filename.ext> ") ;
     scanf("%s", filename) ;
     text(filename) ;
     pointer2 = fopen(filename , "w") ;
     fprintf(pointer2,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f
     %f %f %f %f %f %f",
PRF,AA,T,RT,EA,FF,PL,BL,BW,ST,NF,POWER,AZR,NS,BAZ,BEL,FAN,PJ,GJJ,JR)
    }
    i = 'n' ;
```

```
    }
   }
  }

  h = 1 ;
  settext(&h,&w,&p,&m) ;
  cx = 60.0  ;
  cy =  1.0  ;
  settextclr(&yellow,&black) ;
  movtcurabs(&cx,&cy) ;
  text("Hit any key to continue") ;
  i = getch() ;
  setcolor(&black) ;
  i = getch(12) ;
  clr() ;
  return ;
}
```

```
input2()
{
 FILE *pointer3 ;


 setviewport(&vx1,&vy1,&vx2,&vy2,&border,&back) ;
 pixels = 13 ;
 setlnwidth(&pixels) ;
 setcolor(&red)  ;
 box(&x1,&y1,&x2,&y2);
 cx = 50.0  ;
 cy = 49.0  ;
 movabs(&cx,&cy)      ;
 flood(&liblue)       ;
 setxor(&switsh)      ;
 h = 3 ; w = 2 ;
 p = 0 ; m = 0 ;
 settext(&h,&w,&p,&m) ;
 settextclr(&green,&black) ;
 cx = 30.5 ; cy = 45.0 ;
 movtcurabs(&cx,&cy) ;
 text("INPUT MODULE") ;
 settextclr(&blue,&black) ;
 h = 2 ; w = 1 ;
 settext(&h,&w,&p,&m) ;
 cx = 30.0 ; cy = 40.0 ;
 movtcurabs(&cx,&cy) ;
 text("TRACKING RADAR PARAMETERS") ;
 settextclr(&purple,&black) ;
 cx =  5.0 ; cy = 35.0 ;
 movtcurabs(&cx,&cy) ;
 text("For the explanations of these parameters please ");
 cy = 32.0 ;
 movtcurabs(&cx,&cy) ;
 text("refer to the manual. ") ;

 h = 1 ;
 settext(&h,&w,&p,&m) ;
 cx =  5.0 ; cy = 23.0 ;
 movtcurabs(&cx,&cy) ;
 text("a.EAT   =                    g.CST = ") ;
 cy = 21.0  ;
 movtcurabs(&cx,&cy) ;
 text("b.TAZ   =                    h.NFT = ") ;
 cy = 19.0  ;
 movtcurabs(&cx,&cy) ;
 text("c.GLT   =                    i.BLT = ") ;
 cy = 17.0  ;
 movtcurabs(&cx,&cy) ;
 text("d.TBEL  =                    j.AAT = ") ;
 cy = 15.0  ;
```

```
 movtcurabs(&cx,&cy) ;
 text("e.STT    =                    k.BWT = ") ;
 cy = 13.0    ;
 movtcurabs(&cx,&cy) ;
 text("f.POWT   =                    l.FFT = ") ;
 cy = 11.0 ;
 movtcurabs(&cx,&cy) ;
 text("x.EXIT");
 pointer3 = fopen("track.dfl","r") ;
fscanf(pointer3,"%f   %f  %f  %f  %f  %f  %f  %f  %f   %f   %f
%f",&AAT,&EAT,&BWT,&CST,
 &POWT,&FFT,&GLT,&BLT,&STT,&NFT,&TAZ,&TBEL ) ;
 sprintf(s,"%f", EAT) ;
 cx = 21.0 ; cy = 23.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , CST) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , TAZ) ;
 cx = 21.0 ; cy = 21.0 ;
 movtcurabs(&cx,&cy) ;
 text(s)  ;
 sprintf(s , "%f" , NFT) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , GLT ) ;
 cx = 21.0 ; cy = 19.0 ;
 movtcurabs(&cx,&cy) ;
 text(s)  ;
 sprintf(s , "%f" , BLT) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" ,TBEL) ;
 cx = 21.0 ; cy = 17.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , AAT) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , STT) ;
 cx = 21.0 ; cy = 15.0 ;
 movtcurabs(&cx,&cy) ;
 text(s)  ;
 sprintf(s , "%f" , BWT);
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
```

```
 text(s) ;
 sprintf(s , "%f" , POWT) ;
 cx = 21.0  ; cy = 13.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 sprintf(s , "%f" , FFT) ;
 cx = 65.0 ;
 movtcurabs(&cx,&cy) ;
 text(s) ;
 return   ;
}
```

```
change2()
{
 int  c ;
 static char filename[13] ;
 FILE *pointer4 ;

 h = 2 ;
 w = 1 ;
 p = 0 ;
 m = 0 ;
 settext(&h,&w,&p,&m) ;
 cx = 5.0 ; cy = 29.0 ;
 movtcurabs(&cx,&cy) ;
text("Would you like to change any of these values ? <Y/N>  ")
;
 cx = 93.0 ; cy = 29.0 ;
 movtcurabs(&cx,&cy) ;
 i = getch() ;
 if ( i == 'n' )
  text("No")   ;
 if ( i == 'y' )
 {
  text("Yes") ;
  while ( i  == 'y' )
  {
   h = 1 ;
   settext(&h,&w,&p,&m) ;
   c = getch() ;
   switch(c)
   {
    case 'a' :
    cx = 21.0 ; cy = 23.0 ;
    movtcurabs(&cx,&cy) ;
    sprintf(s ,"%f" , EAT) ;
    text(s) ;
    movtcurabs(&cx,&cy) ;
    scanf("%f" , &EAT) ;
    sprintf(s,"%f", EAT) ;
    text(s)  ;
    break ;
    case 'b' :
    cx = 21.0 ; cy = 21.0 ;
    movtcurabs(&cx,&cy) ;
    sprintf(s ,"%f" , TAZ) ;
    text(s) ;
    movtcurabs(&cx,&cy) ;
    scanf("%f" , &TAZ) ;
    sprintf(s,"%f", TAZ) ;
    text(s)  ;
    break ;
    case 'c' :
```

```
cx = 21.0 ; cy = 19.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , GLT ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &GLT ) ;
sprintf(s,"%f", GLT) ;
text(s)  ;
break ;
case 'd' :
cx = 21.0 ; cy = 17.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , TBEL) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &TBEL) ;
sprintf(s,"%f", TBEL) ;
text(s)  ;
break ;
case 'e' :
cx = 21.0 ; cy = 15.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" ,STT) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &STT);
sprintf(s,"%f",STT) ;
text(s)  ;
break ;
case 'f' :
cx = 21.0 ; cy = 13.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" ,POWT ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &POWT) ;
sprintf(s,"%f", POWT) ;
text(s)  ;
break ;
case 'g' :
cx = 65.0 ; cy = 23.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , CST ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &CST ) ;
sprintf(s,"%f", CST ) ;
text(s)  ;
break ;
case 'h' :
cx = 65.0 ; cy = 21.0 ;
```

```
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , NFT ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &NFT ) ;
sprintf(s,"%f", NFT ) ;
text(s)  ;
break ;
case 'i' :
cx = 65.0 ; cy = 19.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , BLT ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &BLT ) ;
sprintf(s,"%f", BLT ) ;
text(s)  ;
break ;
case 'j' :
cx = 65.0 ; cy = 17.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , AAT ) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &AAT ) ;
sprintf(s,"%f", AAT ) ;
text(s)  ;
break ;
case 'k' :
cx = 65.0 ; cy = 15.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , BWT) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &BWT) ;
sprintf(s,"%f", BWT) ;
text(s)  ;
break ;
case 'l' :
cx = 65.0 ; cy = 13.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s ,"%f" , FFT) ;
text(s) ;
movtcurabs(&cx,&cy) ;
scanf("%f" , &FFT) ;
sprintf(s,"%f", FFT) ;
text(s)  ;
break ;
case 'x' :
h = 2 ; w = 1 ;
settext(&h,&w,&p,&m) ;
```

```
    cx = 10.0 ; cy = 8.0 ;
    movtcurabs(&cx,&cy) ;
    text("DO YOU WISH TO SAVE YOUR DATA YOU JUST ENTERED ?") ;
    i = getch() ;
    if (i == 'y')
    {
     movtcurabs(&cx,&cy) ;
     text("DO YOU WISH TO SAVE YOUR DATA YOU JUST ENTERED ?") ;
     movtcurabs(&cx,&cy) ;
     text("WHAT IS YOUR DATA FILENAME ? <filename.ext> ") ;
     scanf("%s", filename) ;
     text(filename) ;
     pointer4 = fopen(filename , "w") ;
fprintf(pointer4,"%f   %f  %f  %f  %f  %f  %f  %f  %f  %f  %f
%f",AAT,EAT,BWT,
     CST,POWT,FFT,GLT,BLT,STT,NFT,TAZ,TBEL);
    }
    i = 'n' ;
   }
  }
 }
 h = 1 ;
 settext(&h,&w,&p,&m) ;
 cx = 60.0  ;
 cy =  1.0  ;
 settextclr(&yellow,&black) ;
 movtcurabs(&cx,&cy) ;
 text("Hit any key to continue") ;

 i = getch() ;
 setcolor(&black) ;
 clr() ;
 return ;
}
```

```
#include "stdio.h"
#include "math.h"

float x1 = 0.0 , y1 = 0.0  ;
float x2 = 100.0,y2 = 100.0 ;
float v11 = 0.635 , v12 = 0.0 , v13 = 1.0 , v14 = 1.0 ;
float v21 = 0.0 , v22 = 0.0 , v23 = 0.635 , v24 = 1.0 ;
float max_range , lrange , mrange , srange  ;
float dlrange, dmrange, dsrange ;
float cx , cy ;
int back = -1 , border = -1 ;
int h , w , p , m ;
int switsh = 1 ;

int black  = 0  ;
int dblue  = 1  ;
int green  = 2  ;
int lblue  = 3  ;
int red    = 4  ;
int purple = 5  ;
int brown  = 6  ;
int gray   = 7  ;
int dgray  = 8  ;
int blue   = 9  ;
int lgreen = 10 ;
int vlblue = 11 ;
int pink   = 12 ;
int lpurple= 13 ;
int yellow = 14 ;
int white  = 15 ;

float RCPA , RB ;
float xx , yy ;
float XD , YD ;
float xt, yt , as ; /* initial  position of the target */
float zt      ; /* constant altitude of the target */
float U , at ; /* velocity and course of target   */
float R       ; /* distance between ship and target*/

main()
 {
   int mode ;

   setdev("HALOIBME.DEV") ;
   mode = 4   ;
   initgraphics(&mode) ;
   setworld(&x1,&y1,&x2,&y2) ;
   CPA() ;
   Input()  ;
   Opening() ;
   info() ;
```

```
    draw() ;
    detect() ;
    closegraphics() ;
}
```

```
Input()
 {

   printf("\n PLEASE ENTER THE MAXIMUM RANGE DETECTION    ") ;
   scanf("%f" , &max_range ) ;
    printf("\n  PLEASE ENTER THE LONG ,  MEDIUM AND SHORT RANGE
 DISTANCES \n") ;
   scanf("%f%f%f" , &lrange , &mrange , &srange ) ;
   dlrange = 45.0 * lrange / max_range  ;
   dmrange = 45.0 * mrange / max_range  ;
   dsrange = 45.0 * srange / max_range  ;
   setcolor(&black) ;
   clr() ;
   return ;

 }
```

```
Opening()
 {

  float bx , by ;
  int j ;
  int pixels ;
  float radius ;
  float aspect = 0.865 ;

  pixels = 13 ;
  setlnwidth(&pixels) ;
  setcolor(&red)  ;
  box(&x1,&y1,&x2,&y2);
  setviewport(&v11,&v12,&v13,&v14,&border,&back) ;
     setxor(&switsh) ;
     pixels = 1 ;
     setlnwidth(&pixels) ;
     cx = 0.0 ; cy = 0.0 ;
     movabs(&cx,&cy) ;
     cy = 100.0 ;
     lnabs(&cx,&cy) ;
     cx = 50.0  ;
     cy = 98.0  ;
     movabs(&cx,&cy)      ;
     flood(&lblue)       ;
  setviewport(&v21,&v22,&v23,&v24,&border,&back) ;
     setxor(&switsh) ;
     cx = 49.0 ; cy = 49.0 ;
     bx = 51.0 ; by = 51.0 ;
     box(&cx,&cy,&bx,&by)  ;
     setcolor(&yellow) ;
     cx = 1.0 ; cy = 50.0 ;
     movabs(&cx,&cy) ;
     cx = 99.0 ;
     lnabs(&cx,&cy) ;
     cx = 50.0 ; cy = 98.5 ;
     movabs(&cx,&cy) ;
     cy = 1.5 ;
     lnabs(&cx,&cy) ;
     cy =  0.0 ;
     for(j=0;j<19;j++)
     {
      cx = 49.5      ;
      cy = cy + 5.0 ;
      movabs(&cx,&cy) ;
      cx = 50.5      ;
      lnabs(&cx,&cy) ;
     }
     cx = 0.0 ;
     for(j=0;j<19;j++)
     {
```

```
        cx = cx + 5.0 ;
        cy = 49.5     ;
        movabs(&cx,&cy) ;
        cy = 50.5     ;
        lnabs(&cx,&cy) ;
      }
      setcolor(&green) ;
      cx = 50.0 ; cy = 50.0 ;
      movabs(&cx,&cy) ;
      setasp(&aspect) ;
      radius = 45.0 ;
      cir(&radius) ;
      setcolor(&dblue) ;
      movabs(&cx,&cy) ;
      cir(&dlrange) ;
      movabs(&cx,&cy) ;
      cir(&dmrange) ;
      movabs(&cx,&cy) ;
      cir(&dsrange) ;
      h = 1 ; w = 1 ;
      p = 0 ; m = 0 ;
      settext(&h,&w,&p,&m) ;
      settextclr(&green,&black) ;
      cx = 83.0 ; cy = 83.0 ;
      movtcurabs(&cx,&cy) ;
      text("DR") ;
      cx = 50.0 + dlrange ;
      cy = 51.0 ;
      movtcurabs(&cx,&cy) ;
      text("LR") ;
      cx = 50.0 + dmrange ;
      movtcurabs(&cx,&cy) ;
      text("MR") ;
      cx = 50.0 + dsrange ;
      movtcurabs(&cx,&cy) ;
      text("SR") ;

  setviewport(&v11,&v12,&v13,&v14,&border,&back) ;
      setxor(&switsh) ;
      setcolor(&brown) ;
      cx = 0.0 ; cy = 70.0 ;
      movabs(&cx,&cy) ;
      cx = 100.0 ;
      lnabs(&cx,&cy) ;
      cy = 40 ;
      movabs(&cx,&cy) ;
      cx = 0.0 ;
      lnabs(&cx,&cy) ;
      h = 1 ; w = 1 ;
      p = 0 ; m = 0 ;
      settext(&h,&w,&p,&m) ;
```

```
settextclr(&lgreen,&black) ;
cx = 40.0 ; cy = 95.0 ;
movtcurabs(&cx,&cy) ;
text("RADAR") ;
cx = 35.0 ; cy = 67.0 ;
movtcurabs(&cx,&cy) ;
text("MISSILES") ;
cx = 38.0 ; cy = 37.0 ;
movtcurabs(&cx,&cy) ;
text("TARGET") ;
settextclr(&dgray,&black) ;
cx = 18.0 ; cy = 90.0 ;
movtcurabs(&cx,&cy) ;
text("Search Radar active");
cx = 5.0 ; cy = 84.0 ;
settextclr(&pink,&black) ;
movtcurabs(&cx,&cy) ;
text("Frequency :              MHZ") ;
cy = 80.0 ;
movtcurabs(&cx,&cy) ;
text("Max detec :") ;
cx = 28.0 ; cy = 72.0 ;
movtcurabs(&cx,&cy) ;
settextclr(&gray,&black) ;
text(" NO CONTACT ") ;
cx = 5.0 ; cy = 63.0 ;
movtcurabs(&cx,&cy) ;
settextclr(&pink,&black) ;
text("        RANGE    QTY STATUS");
cx = 5.0 ; cy = 58.0 ;
movtcurabs(&cx,&cy) ;
text("Long  :") ;
cy = 54.0 ;
movtcurabs(&cx,&cy) ;
text("Medium:") ;
cy = 50.0 ;
movtcurabs(&cx,&cy) ;
text("Short :") ;
cy = 32.0 ;
movtcurabs(&cx,&cy) ;
text("  ID  :") ;
cy = 29.0 ;
movtcurabs(&cx,&cy) ;
text("   X  :") ;
cy = 26.0 ;
movtcurabs(&cx,&cy) ;
text("   Y  :") ;
cy = 23.0 ;
movtcurabs(&cx,&cy) ;
text("   Z  :") ;
cy = 20.0 ;
```

```
        movtcurabs(&cx,&cy) ;
        text("RANGE :") ;
        cy = 17.0 ;
        movtcurabs(&cx,&cy) ;
        text("  CPA :") ;
        cy = 14.0 ;
        movtcurabs(&cx,&cy) ;
        text("   X  :");
        cy = 11.0 ;
        movtcurabs(&cx,&cy) ;
        text("   Y  :") ;
        cy =  8.0 ;
        movtcurabs(&cx,&cy) ;
        text("Speed :") ;
        cy = 5.0 ;
        movtcurabs(&cx,&cy) ;
        text("Course:") ;
        return   ;
}
```

```
info()
 {
  float freq = 15.5679 ;
  int sqty = 16 , mqty = 8 , lqty = 8 ;
  int i ;
  static char s[14] ;

  setviewport(&v11,&v12,&v13,&v14,&border,&back);
     setxor(&switsh) ;
     settextclr(&gray,&black) ;
      cx = 42.0 ; cy = 84.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%f", freq) ;
      text(s) ;
      cy = 80.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%f",max_range) ;
      text(s) ;
      cx = 29.0 ; cy = 58.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%f",lrange) ;
      text(s) ;
      cx = 65.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%d",lqty) ;
      text(s) ;
      cx = 80.0 ;
      movtcurabs(&cx,&cy) ;
      text("Ready") ;
      cx = 29.0 ; cy = 54.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%f",mrange) ;
      text(s) ;
      cx = 65.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%d",mqty) ;
      text(s) ;
      cx = 80.0 ;
      movtcurabs(&cx,&cy) ;
      text("Ready") ;
      cx = 29.0 ; cy = 50.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%f",srange) ;
      text(s) ;
      cx = 65.0 ;
      movtcurabs(&cx,&cy) ;
      sprintf(s,"%d",sqty) ;
      text(s) ;
      cx = 80.0 ;
      movtcurabs(&cx,&cy) ;
      text("Ready") ;
```

```
    settextclr(&dgray,&black) ;
    cx = 15.0  ;
    cy =  2.0  ;
    movtcurabs(&cx,&cy) ;
    text("Hit any key to continue") ;
    i = getch()  ;
    return ;
}
```

```
CPA()
 {
  float xs, ys ; /* initial  position of the ship   */
  float vs, as ; /* velocity and course of the ship */
  float A      ; /* angle between horizontal and R  */
  float x1, y1 ; /* define ship's velocity vector   */
  float x2, y2 ; /* define target's velocity vector */
  float x , y  ; /* target's relative velocity vect */
  float B      ; /* angle of x-y vector             */
  float C      ; /* angle between R and x-y vector  */
  float CPA    ; /* CPA in X-Y plan                 */
  float AA , BB , CC ;

 printf("\n PLEASE INPUT X,  Y AND Z POSITION OF THE TARGET  :
") ;
  scanf("%f%f%f", &xt, &yt, &zt) ;
  printf("\n PLEASE ENTER ITS SPEED AND COURSE : ") ;
  scanf("%f%f", &U, &at) ;

  printf("\n PLEASE INPUT X AND Y POSITION OF THE SHIP : ") ;
  scanf("%f%f", &xs, &ys) ;
  printf("\n PLEASE ENTER ITS SPEED AND COURSE : ") ;
  scanf("%f%f", &vs, &as);

  at = at * PI / 180.0  ;
  as = as * PI / 180.0   ;
  R = pow((xs-xt)*(xs-xt)+(ys-yt)*(ys-yt) , 0.5)   ;
  A = atan2((yt-ys),(xt-xs))    ;
  x1 = vs * cos(as)             ;
  y1 = vs * sin(as)          ;
  x2 = U  * cos(at)          ;
  y2 = U  * sin(at)          ;
  x  = x1 - x2 + 0.000001 ; /* to avoid division by zero */
  y  = y1 - y2            ;
  B  = atan2(y,x)     ;
  C  = A - B        ;
  if ( C > 1.57079 || C < -1.57079 )
  {
   return ;
  }
  CPA = R * sin(C)    ;
  RCPA= pow((CPA * CPA)+(zt * zt), 0.5 ) ;
  AA = PI/2 - A ;
  CC = PI/2 - C ;
  BB = AA - CC ;
  xx = CPA * sin(BB) + xs ;
  yy = CPA * cos(BB) + ys ;
  RB = B + PI ;
  return  ;
}
```

```
draw()
 {
  float dxx , dyy ;

  setviewport(&v21,&v22,&v23,&v24,&border,&back) ;
   setxor(&switsh) ;
   XD = 50.0 + 45.0 * xt / max_range ;
   YD = 50.0 + 45.0 * yt / max_range ;
   setcolor(&red);
   movabs(&XD,&YD) ;
   dxx = 50.0 + 45.0 * xx / max_range ;
   dyy = 50.0 + 45.0 * yy / max_range ;
   lnabs(&dxx,&dyy) ;
   cx = 50.0 ; cy = 50.0 ;
   lnabs(&cx,&cy) ;
   return ;
}
```

```
detect()
{
  static char s[14] ;
  int i ;
  float atd ;

  setviewport(&v21,&v22,&v23,&v24,&border,&back) ;
   setcolor(&white) ;
   while(R > max_range)
   {
    xt = xt + 10.0 / 3600.0 * U * cos(RB) ;
    yt = yt + 10.0 / 3600.0 * U * sin(RB) ;
    R  = pow(xt * xt + yt * yt + zt * zt , 0.5) ;
    XD = 50.0 + 45.0 * xt / max_range ;
    YD = 50.0 + 45.0 * yt / max_range ;
    ptabs(&XD,&YD) ;
   }
  setviewport(&v11,&v12,&v13,&v14,&border,&back) ;
   settextclr(&gray,&black) ;
   cx = 28.0 ; cy = 72.0 ;
   movtcurabs(&cx,&cy) ;
   text(" NO CONTACT") ;
   m = 1 ;
   settext(&h,&w,&p,&m) ;
   settextclr(&lgreen,&gray);
   cx = 18.0 ;
   movtcurabs(&cx,&cy) ;
   text("WARNING ! CONTACT") ;
   m = 0 ;
   settext(&h,&w,&p,&m) ;
   settextclr(&gray,&black) ;
   cx = 35.0 ; cy = 32.0 ;
   movtcurabs(&cx,&cy) ;
   text("Hostile") ;
   cy = 17.0 ;
   movtcurabs(&cx,&cy) ;
   sprintf(s,"%f",RCPA) ;
   text(s) ;
   cy = 14.0 ;
   movtcurabs(&cx,&cy) ;
   sprintf(s,"%f",xx) ;
   text(s) ;
   cy = 11.0 ;
   movtcurabs(&cx,&cy) ;
   sprintf(s,"%f",yy) ;
   text(s) ;
   cy =  8.0 ;
   movtcurabs(&cx,&cy) ;
   sprintf(s,"%f",U) ;
   text(s) ;
   cy = 5.0 ;
```

```
movtcurabs(&cx,&cy) ;
atd = 180.0 * at / PI ;
sprintf(s,"%f",atd) ;
text(s) ;
settextclr(&lgreen,&black);
cy = 29.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s,"%f",xt) ;
text(s) ;
cy = 26.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s,"%f",yt) ;
text(s) ;
cy = 23.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s,"%f",zt) ;
text(s) ;
cy = 20.0 ;
movtcurabs(&cx,&cy) ;
sprintf(s,"%f",R)   ;
text(s) ;
while(R <= max_range)
{
 setviewport(&v21,&v22,&v23,&v24,&border,&back) ;
  setcolor(&white) ;
  ptabs(&XD,&YD) ;
 setviewport(&v11,&v12,&v13,&v14,&border,&back) ;
  setxor(&switsh) ;
  m = 1 ;
  settext(&h,&w,&p,&m) ;
  settextclr(&lgreen,&gray);
  cx = 18.0 ; cy = 72.0 ;
  movtcurabs(&cx,&cy) ;
  text("WARNING ! CONTACT") ;
  m = 0 ;
  settext(&h,&w,&p,&m) ;
  settextclr(&lgreen,&black) ;
  cx = 35.0 ; cy = 29.0 ;
  movtcurabs(&cx,&cy) ;
  sprintf(s,"%f",xt) ;
  text(s) ;
  xt = xt + 10.0 / 3600.0 * U * cos(RB) ;
  movtcurabs(&cx,&cy) ;
  sprintf(s,"%f",xt) ;
  text(s) ;
  cy = 26.0 ;
  movtcurabs(&cx,&cy) ;
  sprintf(s,"%f",yt) ;
  text(s) ;
  yt = yt + 10.0 / 3600.0 * U * sin(RB) ;
  movtcurabs(&cx,&cy) ;
```

```
  sprintf(s,"%f",yt) ;
  text(s) ;
  cy = 23.0 ;
  movtcurabs(&cx,&cy) ;
  sprintf(s,"%f",zt) ;
  text(s) ;
  movtcurabs(&cx,&cy) ;
  sprintf(s,"%f",zt) ;
  text(s) ;
  cy = 20.0 ;
  movtcurabs(&cx,&cy) ;
  sprintf(s,"%f",R ) ;
  text(s) ;
  R  = pow(xt*xt + yt*yt + zt*zt , 0.5 ) ;
  movtcurabs(&cx,&cy) ;
  sprintf(s,"%f",R) ;
  text(s) ;
  XD = 50.0 + 45.0 * xt / max_range ;
  YD = 50.0 + 45.0 * yt / max_range ;
}
i = getch() ;
cx = 97.0 ; cy = 2.0 ;
movtcurabs(&cx,&cy) ;
i = getch()  ;
return ;
}
```

```
#include "stdio.h"
#include "math.h"
double U , RCPA ;

MAIN()
 {
  CPA() ;
  NUMBER() ;
  exit(0)  ;
}

CPA()
     /* The CPA() function computes the CPA between the */
     /* target and the ship. Two CPA's are computed one */
     /* is the CPA in X-Y plan ( or horizontal plane ), */
     /* the other one TCPA ( true CPA ) is taking  the  */
     /* altitude of the target into account. The CPA is */
     /* converted  into TCPA by changing  from X-Y plan */
     /* to Z plan. The parameters needed for this func- */
     /* tion will be explained when they are declared . */

{
 float xs, ys ; /* initial  position of the ship   */
 float xt, yt ; /* initial  position of the target */
 float zt     ; /* constant altitude of the target */
 float vs, as ; /* velocity and course of the ship */
 float     at ; /* velocity and course of target   */
 float R      ; /* distance between ship and target*/
 float A      ; /* angle between horizontal and R  */
 float x1, y1 ; /* define ship's velocity vector   */
 float x2, y2 ; /* define target's velocity vector */
 float x , y  ; /* target's relative velocity vect */
 float B      ; /* angle of x-y vector             */
 float C      ; /* angle between R and x-y vector  */
 float CPA    ; /* CPA in X-Y plan                          */
 /*float RCPA   ;  CPA with zt component         */

  printf("\n PLEASE INPUT X,  Y AND Z POSITION OF THE TARGET :
") ;
 scanf("%f%f%f", &xt, &yt, &zt) ;
 printf("\n PLEASE ENTER ITS SPEED AND COURSE : ") ;
 scanf("%lf%f", &U, &at) ;

  printf("\n PLEASE INPUT X AND Y POSITION OF THE SHIP : ") ;
 scanf("%f%f", &xs, &ys) ;
 printf("\n PLEASE ENTER ITS SPEED AND COURSE : ") ;
 scanf("%f%f", &vs, &as);

 at = at * PI / 180.0  ;
 printf("\n AT = %f ", at ) ;
 as = as * PI / 180.0   ;
```

```
R = pow((xs-xt)*(xs-xt)+(ys-yt)*(ys-yt) , 0.5)    ;
printf("\n R = %f" , R) ;
A = atan2((yt-ys),(xt-xs))  . ;
printf("\n A = %f " , A) ;
x1 = vs * cos(as)              ;
y1 = vs * sin(as)             ;
x2 = U  * cos(at)             ;
y2 = U  * sin(at)             ;
x  = x1 - x2 + 0.000001 ; /* to avoid division by zero */
y  = y1 - y2                  ;
printf("\n X = %f  , Y = %f ", x,y ) ;
B  = atan2(y,x)      ;
printf("\n B = %f ", B ) ;
C  = A - B          ;
printf("\n C = %f ", C ) ;
if ( C > 1.57079 || C < -1.57079 )
{
 printf("\n NO CPA ! ") ;
 return ;
}
CPA = R * sin(C)     ;
RCPA= pow((CPA * CPA)+(zt * zt), 0.5 ) ;
printf("\n THE CPA IS : %f", CPA );
printf("\n THE TRUE CPA IS : %lf", RCPA ) ;
return  ;
}
```

```
NUMBER()
 {
 double RI[10] , RT[10] , TA[10] , LA[10]  ;
 double NPHI , PHI , NR , NL , DIST , T , V , RX ;
 double TEMP , RN ;
 double PK[10] , TPK = 1.0 ;
 int  N  , i  ;

  printf("\n PLEASE ENTER THE MAXIMUM THEN THE MINIMUM MISSILE
RANGE");
 scanf("%lf%lf" , &RX , &RN );
 if (RCPA > RX)
 {
  printf("\n THE TARGET IS BEYOND THE MISSILE'S RANGE !")  ;
  return  ;
 }
 printf("\n ENTER THE SPEED OF MISSILE " ) ;
 scanf("%lf" ,  &V ) ;
 printf("\n ENTER THE DELAY TIME T ") ;
 scanf("%lf" , &T ) ;

 RI[0] = RX   ;
 TA[0] = 0   ;
 N     = 0  ;
 TEMP  = RCPA/RI[0] ;
 NPHI  = asin(TEMP)       ;
 printf("\n RCPA = %lf , T = %lf , U = %lf " , RCPA, T, U ) ;
 DIST  = T * U  ;
 TEMP  = (DIST*DIST + RI[0]*RI[0] - 2*DIST*RI[0]*cos(NPHI)) ;
 NR    = pow(TEMP , 0.5 )  ;
 TEMP  = DIST*RCPA/(NR*RI[0]) ;
 NL    = asin(TEMP) ;
  printf("\n NPHI = %lf     , DIST = %lf     , NR = %lf    , NL =
%lf     " , NPHI , DIST , NR , NL )  ;
 TA[1] = NPHI + NL  ;
 TEMP  = U/V*sin(TA[1])         ;
 LA[1] = asin(TEMP)     ;
 PHI   = PI - TA[1] - LA[1]    ;
 RI[1] = RCPA / sin(PHI)       ;
 RT[1] = U/V*RI[1]             ;
 N     = 1                   ;

 while (RI[N] <= RX )
 {
  if (RI[N] < RN)
  {
   TEMP = RCPA / RN          ;
   NPHI = PI - asin(TEMP)   ;
   TEMP = DIST*DIST + RN*RN - 2*DIST*RN*cos(NPHI)   ;
   NR   = pow(TEMP , 0.5)     ;
   TEMP = DIST*RCPA/(NR*RN)   ;
```

```
    NL    = asin(TEMP)          ;
    TA[N] = NPHI + NL    ;
    TEMP  = U/V*sin(TA[N]) ;
    LA[N] = asin(TEMP)   ;
    PHI   = PI - TA[N] - LA[N]     ;
    RI[N] = RCPA / sin(PHI)        ;
    RT[N] = U/V*RI[N]   ;
    if (RI[N] >= RX )
    {
     printf("\n NUMBER OF MISSILES : %d ", N )   ;
     return   ;
    }
   }
   NPHI   = PI - PHI       ;
   TEMP   = (DIST*DIST + RI[N]*RI[N] - 2*DIST*RI[N]*cos(NPHI))   ;
   NR     = pow(TEMP , 0.5 ) ;
   TEMP   = DIST*RCPA/(NR*RI[N])   ;
   NL     = asin(TEMP)      ;
   N      =  N + 1    ;
   TA[N] = NPHI + NL    ;
   TEMP   = U/V*sin(TA[N]) ;
   LA[N] = asin(TEMP)   ;
   PHI    = PI - TA[N] - LA[N]     ;
   RI[N] = RCPA / sin(PHI)        ;
   RT[N] = U/V*RI[N]   ;
 }
 for (i=0;i<N;i++)
 {
  PK[i] = 1.233333 - RI[i]/24.0;
  TPK   = TPK * ( 1.0 - PK[i] ) ;
 }
 TPK  = 1.0 - TPK     ;
 printf("\n RCPA = %lf , RX = %lf , RN = %lf ", RCPA,RX,RN ) ;
 for (i=0;i<N;i++)
  printf("\nRI[%d]=%lf,TA[%d]=%lf,PK[%d]=%lf"
,i,RI[i],i,TA[i],i,PK[i]);
 printf("\n THE TOTAL PROBABILITY OF KILL IS %lf ", TPK );
  printf("\n THE NUMBER OF MISSILES THAT CAN BE FIRED :  %d ",
N );
 return   ;
}
```

LIST OF REFERENCES

1. Montero, C., A Radar Modal for ISEAS, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1986.

2. Antonnio, D., A Missile Flyout Modal for ISEAS, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1986.

3. Naval Operations Analysis, 2nd ed., p. 215, Naval Institute Press, 1977.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Professor Robert E. Ball Code 67Bp Naval Postgraduate School Monterey, California 93943-5000	1
4.	Jawad Mamou 30 Blvd. Emile Zola Casablanca 01 MOROCCO	2
5.	Inspecteur de la Marine Royale Inspection de la Marine Royale Rabat MOROCCO	2

22108?

Thesis
M27848 Mamou
c.1 Computer simulation of an anti-air operation in the Combat Information Center.